JULY 1988 TRAC - F - TM - 0988

ACN 16306

FORT LEAVENWORTH IMPROVED KELLNER (FLIK) GRAPHICS INTERFACE: GENERAL OVERVIEW

Fort Leavenworth

US ARMY
TRADOC ANALYSIS COMMAND-FORT LEAVENWORTH
(TRAC-FLVN)
OPERATIONS DIRECTORATE
FORT LEAVENWORTH, KANSAS 66027

Technical Memorandum TRAC-F-TM-0988
July 1988

TRADOC Analysis Command-Fort Leavenworth (TRAC-FLVN)
Operations Directorate, Technology Applications Branch
Fort Leavenworth, Kansas 66027-5200

FORT LEAVENWORTH IMPROVED KELLNER (FLIK)

GRAPHICS INTERFACE: GENERAL OVERVIEW

by

Tim Daniels

ACN 16306

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TRAC-F-TM-0988	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION TRAC-FLVN; 6b. OFFICE SYMBOL (If applicable) ATRC-FOC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Director, TRAC-FLVN ATTN: ATRC-FOC-T) Fort Leavenworth, KS 66027-5200	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Fort Leavenworth Improved Kellner (FLIK) Graphics Interface General Overview

12. PERSONAL AUTHOR(S)
Timothy Daniels

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Jan 87 TO Oct 88	1988 October	76

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper presents a survey of the Fort Leavenworth Improved Kellner (FLIK) graphics interface system. The survey covers the FLIK developers, the origins of FLIK, how to use FLIK, what problems it solves for applications, problems that remain, enhancements over previous interface packages, and a development history. This paper also contains appendices with more detailed information about some topics discussed briefly within the survey.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code); 22c. OFFICE SYMBOL

ACKNOWLEDGEMENTS

Al Kellner, TRADOC Analysis Command, White Sands Missile Range, (TRAC-WSMR) wrote the Kellner graphics packages. He built a thorough, well designed graphics package. Howard de St. Germain, TRAC-WSMR, made changes to at least one version of the Kellner package to tailor it to certain Ramtek hardware.

The Technology Applications Branch, TRADOC Analysis Command, Fort Leavenworth (TRAC-FLVN), created the Fort Leavenworth Improved Kellner (FLIK) package. Tim Daniels did the majority of the effort in merging Kellner code into FLIK, writing the FLIK station package, OMNI, and the display list software. Pete Kaeding also worked on development of FLIK modules.

Several people at TRAC-FLVN and at TRAC-WSMR extensively tested FLIK in many varied environments:

o VIC input preprocessor (VIP): Pete Kaeding, Mike Chenault, Jim Lankford and Jim Williams.

o The Fort Lewis Weapons-effects package: Pete Kaeding.

o Vector-In-Commander (VIC) version 1.2 interactive game: Mike Hannon.

o VIC version 1.2 PLAYBACK: Pete Kaeding, Mike Hannon, and Terry Gach.

o Graph-Cap package (A pop-up/pull-down menuing and point-data display systems, by Pete Kaeding): Pete Kaeding.

o Color slide package for interactive color slide generation, by Mike Chenault: Mike Chenault.

TABLE OF CONTENTS

ABSTRACT

This paper presents a survey of the Fort Leavenworth Improved Kellner (FLIK) graphics interface system. The survey covers the FLIK developers, the origins of FLIK, how to use FLIK, what problems it solves for applications, problems that remain, enhancements over previous interface packages, and a development history. This paper also contains appendices with more detailed information about some topics discussed briefly within the survey.

#1273

FORT LEAVENWORTH IMPROVED KELLNER (FLIK) GRAPHICS INTERFACE: GENERAL OVERVIEW

1. Introduction. This document explains the development of the FLIK graphics package and how it impacts on other efforts of TRADOC Analysis Command (TRAC) and other Department of Defense (DOD) installations.

a. FLIK is a creation of the Technology Applications Branch (TAB), Computer Systems Division, TRADOC Analysis Command, Fort Leavenworth (TRAC-FLVN). The FLIK graphics interface software package facilitates an application's use of Ramtek graphics systems by converting user graphics requirements to host commands for the Ramtek. Ramtek systems contain a graphics processor that converts host commands into graphics data stored in a bitmap, a large random-access memory (RAM). That bitmap forms a three-dimensional Cartesian coordinate system. A video board maps all memory bits at the same (X,Y) coordinate into a unique pixel, a specific point on the monitor screen surface. The size of the bitmap is exactly sufficient for the video board to fill the entire screen with a display of these pixels. The Z coordinate splits the bitmap into bit-planes. Each bit-plane spans the entire screen but has only one bit per pixel. The video board can combine data from several bit-planes to control each pixel's color and/or to overlay on the screen a picture stored in one set of bit-planes over a picture stored in another set (e.g.: symbols over a terrain map). User applications such as battle simulations, pre- and post-processors, terrain map displays, slide generation packages, and satellite data display systems all benefit from graphics packages such as FLIK.

b. Before FLIK, several other graphics interface packages throughout DOD had the same purpose as FLIK. Al Kellner at TRADOC Analysis Command, White Sands Missile Range (TRAC-WSMR) is the exclusive author of the Kellner graphics package. Each Kellner package is thorough in the exercise of the Ramtek graphics and interactive capabilities. Due to their clean design and wide applicability, Kellner packages are used at several DOD schools.

c. Each Kellner package was tailored to a specific VAX/Ramtek hardware configuration. The package made only partial adjustments for the specific Ramtek hardware used. As a result, the Kellner packages require the user to tailor the applications software to fit a certain Kellner package and associated hardware configuration.

(1) The tailoring requirement for each Kellner package became a problem for some users because it locked their applications into a certain type of Ramtek hardware, making them nontransportable. Nontransportability has been a technical

impediment to software exchanges among various DOD schools and centers. Had there been only one type of Ramtek requiring a single Kellner package, transportability would not been a problem.

(2) FLIK makes virtually all adjustments required by varying Ramtek hardware configurations. It leaves the application virtually free of any such requirements and permits DOD application products to be exchanged among hardware suites and installations.

(a) Transportability is the main benefit of the FLIK package for Kellner users. TAB has even combined applications made for different Ramteks into a single executable and has successfully run that combination with several Ramtek suites at TRAC-FLVN.

(b) For example, we can now run the Vector-in-Commander (VIC) battle simulation version 1.2 on all Ramtek hardware at TRAC-FLVN; before it ran only on TRAC-WSMR-type Ramteks. Also, the VIC Input Processor (VIP) now runs on all Ramteks at TRAC-WSMR and TRAC-FLVN. The Fort Lewis weapons-effects software (designed for a different Ramtek than was VIP) now runs in a combined executable with VIP on all Ramteks at TRAC-FLVN. The VIC playback software now runs on virtually all hardware at TRAC-FLVN. CASTFOREM from TRAC-WSMR's MicroVax now draws graphics at TRAC-FLVN on the quite different Ramteks connected to the VAX 11/785 machines.

d. FLIK is a fusion and an upgrade of several Kellner graphics packages. It contains all the functions of the older Kellner packages, virtually isolates the user from Ramtek dependency, and provides several additional features. FLIK has evolved for the last 18 months, and its use is becoming more widespread. FLIK is used at TRAC-WSMR, and at Combat Analysis Agency (CAA). TRAC-WSMR plans to distribute FLIK with its latest release of VIP to approximately 35 users DOD-wide.

2. Problems Solved. Previously, the user had to tailor application software to solve problems resulting from varying Ramtek hardware parameters. This resulted in a separate set or subset of applications software for each possible type of Ramtek configuration. One solution was to make the application "smart" by branching on a Ramtek "type," entered by the user at run time. This made the application vulnerable to additional "types" of Ramtek hardware. In any case, the application had to link to the correct graphics library specially tailored to one type of Ramtek hardware. This always resulted in separate executables, one for each Ramtek with which the user planned to run. FLIK has only one such library to link to, intended for all types of Ramtek

systems. With FLIK, the user has virtual freedom from Ramtek hardware considerations. As a result, the user does not need to keep multiple sets of applications, nor branch frequently based on hardware parameters. FLIK provides the following solutions:

a. Linking. The user has a single link procedure not dependent on Ramtek hardware.

b. Image resolution. The user has virtual freedom from Ramtek resolution (low or high), except during image transfer. This permits a user to write most application code having integer coordinates and text sizes based on one Ramtek resolution, then compile and link with the FLIK package. FLIK lets the user run the resulting single executable application program on Ramtek hardware having either resolution, low or high. However, during image transfers, FLIK does not adjust two resolution-dependent parameters. It leaves those adjustments for the user to do. These parameters are the refresh memory coordinates for the opposite corners of the rectangular transfer area, and the count of the number of pixels to transfer to the actual Ramtek resolution. FLIK expects the application to branch on the image resolution when determining those values.

c. Color format. Ramtek hardware uses three digital to analog converters (DACs) to output the red/green/blue color-shading combinations to each Ramtek monitor screen. Ramtek Corporation can configure its hardware with sets of either four or eight bit DACs. The user provides a list of color definitions for the shadings desired. Each color definition in that list is the input values to the red/green/blue DACs, represented as either three four-bit values, or three eight-bit values. This would make the host color list format hardware dependent. But, with FLIK, the user's color list has total freedom from such dependencies. That allows one executable of an application that defines colors in one of the two DAC size formats to run equally well with Ramteks having either DAC size.

d. Station-selection parameters. The user has total freedom from identifying and referencing the tablets, cursors, and color tables associated with the Ramtek station on which an application runs. If an application chooses an incorrect peripheral or color table, FLIK prints an error message, finds the correct peripheral, and proceeds.

e. Color levels. Applications software that defines color overlay levels must branch on the number of Ramtek bit-planes available. FLIK provides this information at run time. FLIK frees the user from deciding the association between color levels and Ramtek bit-planes and memory control processors (MCPs).

f. Special video boards. FLIK makes the type of video board transparent to the user. The user need not do any special initialization of the newer V6 boards. For the specialized V6B board there are FLIK options available to an application to take advantage of the colored blinking cursor options.

g. Image pixel-bits. At run time, FLIK provides to the application software information on the number of bit-planes available at the current station. Any application that manipulates images must branch on that information when defining the image pixel buffer size or contents. Also, FLIK establishes internally a default relation between host pixel bits and Ramtek bit-planes at the current station. Using that relation, FLIK frees the application software from the details involved during image transfer of pixels between host memory buffers and Ramtek bit-planes. When the user defines a list of color levels FLIK allocates those color levels to certain Ramtek bit-planes, and changes its internal relation between host pixel bits and Ramtek bit-planes to include only those Ramtek bit-planes used.

3. Converting from Kellner to FLIK. Converting an application from Kellner to FLIK requires a few minor code changes in the application program and a minor change in the linking procedure. Even though these changes are minor, they are essential.

a. User expectations versus Ramtek hardware. One of the main services FLIK provides is the reconciliation of a mismatch between an application written for one Ramtek suite and the Ramtek suite currently in use. This allows portability of graphics applications among Ramtek suites. To solve this potential mismatch, FLIK requires the type of Ramtek the application was written for (user expectations) and the type of Ramtek in use. The latter information comes from the station file and the Ramtek itself. The former comes from the user during FLIK initialization.

b. Initialization. All applications switching from Kellner to FLIK graphics packages must change their argument list in the call to KRMINIT.

(1) The Kellner KRMINIT call was:

```
CALL KRMINIT (IDUM, ICRT)
```

where IDUM did nothing and ICRT was the integer station number (greater than or equal to one).

(2) The FLIK KRMINIT call is:

```
CALL KRMINIT (IXRES, IYRES, IDAC, ICRT)
```

where IXRES and IYRES are the "expected" Ramtek resolution (X and Y), IDAC is the "expected" DAC size, and ICRT is the integer station number, as with the Kellner KRMINIT.

(3) The "expected" Ramtek resolution and DAC size describe the type of Ramtek the application was originally written for when used with the Kellner packages. FLIK requires that information to later adjust integer coordinates, text size, and color formats.

c. Combining applications. Ignore this when working with individual applications packages. It is a FLIK enhancement that allows two or more graphics applications originally written for different Ramtek hardware to be combined into one shell. As the shell transfers control among the various merged packages, it resets the Ramtek "expectations" known by FLIK to match those of the current application. You initialize only once with KRMINIT, when some application under the shell first needs a Ramtek. For every change to a different application within the shell, you reset the Ramtek "expectations" with a call to KUSERMODE. This concept appears in the following code illustration.

```
      PROGRAM SHELL
CC  This program includes applications
CC  software built originally for
CC  different Ramtek suites.
      IXRES1 = 640
      IYRES1 = 512
      IDAC1 = 4
CC
      IXRES2 = 1280
      IYRES2 = 1024
      IDAC2 = 8
CC
      IXRES3 = 640
      IYRES3 = 512
      IDAC3 = 8
CC           FLIK initialization, one time only.
      CALL KRMINIT (IXRES1, IYRES1, IDAC1, IUNCRTL)
      CALL APPLIC1
CC           Notify FLIK that software to follow
CC           was originally written for high res
CC           with eight-bit DACs.
      CALL KUSERMODE (IXRES2, IYRES2, IDAC2)
      CALL APPLIC2
CC            Notify FLIK that software to follow
CC            was originally written for low res
CC            with eight-bit DACs.
      CALL KUSERMODE (IXRES3, IYRES3, IDAC3)
      CALL APPLIC3
CC             Notify FLIK that software to follow
CC              was originally written for low res
CC              with four-bit DACs.
      CALL KUSERMODE (IXRES1, IYRES1, IDAC1)
      CALL APPLIC1
CC
      STOP
      END
```

Note: The three arguments for KUSERMODE are just like the first three arguments in KRMINIT: the expected Ramtek resolution (X and Y) and the expected DAC size.

d. Tablet inputs. FLIK normally will only return from a tablet read when the user pushes a button. Some applications using the Kellner package rely on an immediate return so they can set up a polling loop. This loop will fail in FLIK. To make it work again you must change the old Kellner call to KRTABBUT or KRTABST inside the loop to a FLIK call to either KRTABBUTSEC or KRTABSTSEC, or KTABLET (the latter is easiest.) The modules KRTABBUT and KRTABST now work only in nonpolling situations, and do so more efficiently than before. Refer to Appendix A for an example of changing an old application to still poll with FLIK.

e. Image transfers. Applications using the old Kellner packages' image transfer modules KWIMAGE and KRIMAGE supplied a first argument that was not used by those modules. The documented intent of the first argument was to set the image transfer mode as follows:

```
C    IHMODE:  = 0,  PLANES 0-15 ARE TRANSFERRED  C
C             = 1,  PLANES 0-7  ARE TRANSFERRED  C
C             = 2,  PLANES 8-15 ARE TRANSFERRED  C
```

(1) In tailoring Kellner packages to a Ramtek hardware configuration, the implementers of Kellner disabled this first argument. Internal to the Kellner software, they defined a hard-wired set of image transfer modes which replaced the stubbed-out argument. There was one such mode hard-wired for each possible station on that Ramtek configuration. At run time, during initialization, the user would select a station. Later, when transfering images, the hard-wired mode for that station would determine the correct bit-planes to participate in the image transfer to or from that already-selected station. To match the Ramtek hardware and the Kellner package used, the users had to write their application to manipulate images having either eight or 16 bits per image pixel.

(2) With FLIK, the situation has changed. The first argument in the call to KWIMAGE and KRIMAGE now functions as documented. It does two things: it describes the pixel format of the users' image data (either eight or 16 bits per pixel) and,

if eight bits per pixel, it has the potential to control which bit-planes of a station participate in the image transfer (either lower eight or upper eight bit-planes.)

(3) When changing old applications from Kellner to FLIK, you must check all the calls to KWIMAGE and KRIMAGE. Be sure that the first argument, now active, fits the real situation in the software. Ensure that applications working with 16-bit images always has a zero as the first argument in these calls. Ensure that applications working with eight bit images always have either a one or two as the first argument in these calls. (Usually in the latter case you will use a one, not a two.) In most Kellner packages since the first argument was disabled, users rarely bothered to route their images to the upper eight of a set of 16 planes available to one station. It is only very recently that any Kellner applications have been used on Ramteks where 16 planes are available. Those Ramteks have the new V6B video boards.

(4) In some cases, Kellner implementers added extra modules, such as KWIMAGE_8600_V and KRIMAGE_8600_V, that function similarly to KWIMAGE and KRIMAGE. By calling one of these added modules, a Kellner user could choose between eight and 16 bit images. When changing such software to work with FLIK, you must change all calls back to the original KWIMAGE and KRIMAGE but set the first argument correctly. See Appendix B for an example of such an application program change.

f. Correction of application errors. Applications software usually contains one erroneous assumption and one error pertaining to tablet input.

(1) Erroneous assumption. Often when users work with tablet and cursor input, they rely on the cursor's "enter" flag when their Ramtek has only tablets, and not cursor controllers (such as joystick or trackball) connected. There is also a Ramtek mouse available, but we are not certain how Ramtek treats it.

(a) The Ramtek firmware is such that this cursor "enter" flag never lights up when a tablet puck button is pushed. Cursor coordinates follow the puck, but the cursor "enter" flag does not. Only when your Ramtek has a cursor controller instead of a tablet will the cursor "enter" flag be valid. We challenge anyone to find this fact in the Ramtek Software Reference Manual.

(b) In Appendix A, we note an example of this same erroneous assumption. It appears in the line of code that assigned the IENTER value returned from KRCURS call to the IBUTN value.

(2) Code error. There is an error in application code that has proliferated over the years. It does not appear when using Kellner but makes the tablet go dead when using FLIK. The error was in setting the conditions under which a tablet interrupt occurs. Kellner software does not rely on tablet interrupts, so the application error never had an effect. FLIK does rely on tablet interrupts, and the tablet will fail to respond after the application executes the error. The error is when an application calls KWTABST with a status of ISTAT which is set to a hex value usually of '101', or '201', '301', '401', '501', '601', or '701', (usually it is a '201' or '701'). There is a missing zero in the hex constant. The left-most digit should be the first of FOUR digits, not three. This error means that no tablet-status bits are set to cause a Ramtek interrupt as the result of a button press or button release, or puck movement while one or more buttons are pressed. You must check the ISTAT value in all calls to KWTABST when converting old applications from Kellner to FLIK. In such cases, our experience has been that when a tablet goes dead, this erroneous ISTAT constant invariably has been the cause.

4. <u>Lingering hardware considerations.</u> There are still a few hardware-related considerations that the applications program must prepare for. Most of these should be in the form of branches in the application code. The application code can still remain transportable by getting Ramtek information from FLIK at run time, then branching on that information.

a. Color-level limitations. When defining color levels with FLIK module KOVRDEF, the total number of Ramtek planes assigned must stay at or below the number of planes available on the current Ramtek station in use. An application can get this information by calling module RMSTN_PIXBITS in the station software. If the user tries to use too many planes, FLIK will print an error message. For example, if the current Ramtek station has eight bit-planes and a user asks for two overlay levels with five planes each (total of 10 planes) this is an error. See Appendix C for an example of how to get the number of available planes.

b. Image pixel-bits limitations. The user should be aware that in transferring images, FLIK will only transfer, at maximum, as many bits per host pixel as there are Ramtek bit-planes available at the current station. However, the user can elect to transfer fewer bits per pixel than the maximum. For example, if an application set aside 16 bits per pixel, and the current Ramtek station had 10 bit-planes available, and the user wrote an image using a HOST-MODE=0 (16 bits per pixel) then FLIK would only transfer the lower 10 bits in each host word to the correct Ramtek bit-planes. If the application instead set aside eight bits per pixel and the current Ramtek station still had 10 bit-planes available, and the application wrote images with a HOST-MODE=1 (eight bits per pixel) then FLIK would only transfer the lower eight bits in each host word to the correct Ramtek bit-planes.

c. Actual image size. Image transfers involve rectangular areas of refresh memory, defined by two opposite corners. There are two levels at which to represent image areas--abstract (logical), and physical (actual). A logical representation defines opposite corners proportional to the entire view. An actual representation defines opposite corners in actual pixel addresses in the refresh memory available on the Ramtek currently in use. With images of the same actual physical pixel size transferred into both low and high resolution Ramteks, the viewed image on the low resolution system spans twice the proportional width and height of the total screen, doubling the logical abstract width and height of the low resolution image. Conversely, with images of the same logical abstract proportional size on both a low and a high resolution screen, the stored image in the high resolution system must span twice the pixel width and height in refresh memory, doubling the actual physical width and height of the high resolution image. Applications often need to maintain the same proportional abstract logical image sizes, regardless of system resolution. See Appendix D for an example of making an application do this by adjusting the actual image size.

(1) The Kellner packages and FLIK force users to define their image transfer areas with actual refresh memory pixel dimensions. Therefore, for an application to correctly transfer images with either low or high Ramtek resolution, that application must make allowances for that resolution. Such applications have several resolution-dependent factors on the host side that FLIK cannot control, including the proper sized buffer space and, if moving pictures to and from disk, a suitable disk file and record format.

(2) Therefore, FLIK makes the refresh memory resolution size available to the application via a call to KQWIN. After calling KQWIN, the application can use the returned resolution data when setting the following values:

o The actual location and size of the rectangular transfer area.

o The choice of image buffer by size.

o The disk file and record formats for moving images to and from host disk.

5. <u>Higher efficiency.</u> The FLIK graphics package is more efficient than the original Kellner graphics package in the following respects.

a. Tablet input. With FLIK, the user may now eliminate all unnecessary Ramtek tablet polling. Previously, under all conditions, a tablet read would always return immediately to the application, even when no puck button was pushed. Most of the time this is undesirable, since the application expects a button-push. Now, the Ramtek normally returns to the application only when a button is pushed. Note that FLIK still provides polling capabilities for special cases such as pull-down menus. This nonpolling reduces the system overhead on the host computer. See Appendix A for additional information.

b. Faster imaging. FLIK provides two modules for writing images to the Ramtek: the normal Kellner write image and a write image with event flag. Write image with event flag only starts an image transfer from host to the Ramtek but does not wait as usual for the transfer to complete. Instead, it returns immediately to the application program. The application must do several write images to fill a screen. When moving a full screen picture from disk to Ramtek, the application can use a double buffering technique on these several partial screens of image. This means that after reading the first partial picture from disk, the application can start the write image of the current partial picture and, while that write takes place, read the next partial picture from disk. This almost halves the time to display images.

c. Multistation imaging. TAB at TRAC-FLVN has identified a problem with image displays. It occurs with both Kellner and FLIK when each of two simultaneously running processes has attached to different Ramtek stations driven by the same Ramtek controller. Under those conditions, if both processes attempt to move images from disk back to their respective Ramtek stations, neither picture transfer completes. To solve this problem, we have provided a new FLIK module called KWIMAGEEFN. Changing from KWIMAGE to KWIMAGEEFN requires 11 words at the front of the image buffer and a call to KWAIT afterward. See appendix E for details.

6. <u>New features.</u> FLIK not only removes hardware-related burdens from the user application and provides higher efficiency operations, but also makes available some major new features.

a. Help libraries. For all the major sections of FLIK software, we now have VAX VMS interactive HELP libraries containing documentation on all the user-callable modules available. These libraries were automatically generated from the documentation in the software itself. Some additional documentation has been added in the library for FLIK for clarification. To use the libraries, you enter HELP_FLIK, HELP_DL, HELP_RMSTN, and HELP_OMNI.

(1) HELP_FLIK gives you a library describing all the FLIK modules, arranged into subsets by their general function, such as initialization, vector drawing, color control, display lists, etc.

(2) HELP_DL gives you access to information on the FLIK display list software (named FLIK-DL), including an overview, examples, and module descriptions. FLIK-DL software lets a user prepare display lists, save them on disk files, load them in the Ramtek, and call them.

(3) HELP_RMSTN gives you a library describing the Ramtek station software. This software builds, maintains, and provides user access to a data structure that describes the available Ramtek stations connected to a given processor. FLIK relies heavily on this software for information that enables FLIK to properly relate user applications and existing hardware. This same RMSTN package provides applications with similar information.

(4) HELP_OMNI gives you a help library for the TRAC-FLVN OMNI program. You run OMNI on-line against a user-edited input disk file containing Ramtek instruction in a mnemonic format (as seen in the Ramtek Software Instruction Manual and the Reference Manual). OMNI converts those mnemonics to actual Ramtek code, and sends that code to the Ramtek controller you specify. Use OMNI to research the exact behavior of a Ramtek system and to try examples from the Ramtek course training manual or the Ramtek reference manual.

b. Metafiles. If you select this option, FLIK builds a metafile containing information sent by the application to 21 of the most-used FLIK graphics modules. Later on, the user can run this file through a FLIK metafile display program that will redraw the picture saved on the metafile. See appendix F for operating instructions.

c. Display list functions. FLIK now provides display-list capabilities to applications software. Display lists are programs in Ramtek code format that reside in Ramtek memory. Their usual purpose is to draw graphics, but they can also branch, call, loop, load and store data, and interact with the user through tablet or keyboard. A display list runs when it is called. The host can call display lists and DISPLAY LISTS CAN CALL OTHER DISPLAY LISTS.

(1) Previously, the inclusion of calls, branches, and load-store operations into display list programs required meticulous handling of absolute Ramtek addresses. This was similar to writing in machine code on an old computer that had no assembler or higher-level language compiler. Any attempt to change or augment display list code or to combine several display lists met with a complete review and/or revision of all the absolute addresses contained therein. Needless to say, this usually discouraged all but the most determined user from incorporation of display lists.

(2) We feel that FLIK will make display lists more attractive to the user. FLIK now provides four basic display-list operations: define and compile, link, send, and call. These operations must occur in order but not necessarily under control of the same application. In addition, for display lists that use branches, calls, and load-store operations, FLIK entirely frees the user from all previous burdens of absolute Ramtek addressing. The user works with symbolic names for such addresses. There are three types of symbolic names: local, common, and external. All these symbols are relocatable. This means that the symbolic names the user sees remain constant, while the absolute addresses the user does not see change when the user changes or adds to display list code, or when the user links combinations of various display lists for use together. See appendix G for further explanation and examples.

d. Graphics training. FLIK has an associated package called OMNI that helps you learn about the Ramtek system by trying the examples presented in the Ramtek Systems Software training manual. These examples contain instruction and operand mnemonics, comments, and several types of constants--integer, hexadecimal, and American Standard Code for Information Interchange (ASCII). You enter these examples on an ASCII file on the host, and give that file to OMNI. OMNI translates its contents into Ramtek instructions and sends them to the Ramtek you select.

(1) OMNI, as well as an application using FLIK, can build and use display lists. In the past, OMNI-like graphics training packages provided a primitive environment in which the user worked with absolute Ramtek addresses for any Ramtek memory references, such as load-store, branching, and calls. Now, the new OMNI gives you the option to use symbolic names for those same memory references. This new option mirrors the FLIK display list capabilities. Just as with FLIK, OMNI has four basic display-list operations: define and compile, link, send, and call.

(2) OMNI's display-list object files have the same format as the FLIK display-list object files. Thus, you can link together display lists made by both OMNI and FLIK. You can also call with OMNI display lists made by FLIK, and you can call with FLIK display lists made by OMNI.

7. Known FLIK Limitations. While developing and testing FLIK, we identified what we consider to be limitations.

a. Metafile capability is only in 21 modules. Those modules all draw graphics with integer coordinates. None of the virtual-coordinate graphics have metafile capability yet.

b. The switchable debug facilities are only in a very few modules.

c. The user still must deal with the number of planes available on a station, and with the imaging resolution. FLIK contains no smarts to reduce the number of colors per level to fit on the current station assigned.

d. FLIK has no capability to enable/disable the display of data loaded into refresh memory in selected colors on selected levels, or selected entire levels. However, the TAB design of the color controls in FLIK will easily accomodate this feature. For example, with an eight plane Ramtek, you might load point-data containing eight types of vegetation/standing water in the lower three planes, two kinds of roads and one type of river in the two planes above that, temporary information in the next plane, and military unit symbols for up to three forces in the top two planes. You would define colors and levels accordingly. Then, with this as yet unimplemented capability, you could IMMEDIATELY make invisible all military forces, all units in a specific force, all roads and rivers, selected types of roads, rivers only, the temporary data, or the vegetation background. You could later IMMEDIATELY make visible again that which you had made invisible and do so WITHOUT the host taking time to redraw it. While that data was invisible, the host could still change it in Ramtek refresh. When that data became visible again you would IMMEDIATELY see the updated information.

e. The original KWIMAGE will still blow up when two processes simultaneously write images to separate stations on the same Ramtek controller. An application can convert to using KWIMAGEEFN as a replacement, but users of the old KWIMAGE will still have problems.

f. There is no pixel-formatter emulation in FLIK for hardware having no pixel formatters. There is no reverse pixel-formatter emulation for reading back images split into two or more sets of planes. Also, we have not yet tested FLIK's ability to handle more than one pixel segment per image word.

g. FLIK resolves problems of hardware dependency while building display lists. This makes the display list code hardware dependent, so the display lists made for one Ramtek configuration will not work on another. The current simple solution is to keep a set of display lists for each configuration you use. Two other more complex approaches to this solution are mentioned in appendix G.

8. <u>Development order.</u> The development of the FLIK graphics package from the Kellner graphics package has had several iterations.

a. FLIK Version 1.0. This version incorporated the following software: Kellner TRAC-WSMR low resolution, TRAC-WSMR high resolution, and Fort Lewis high resolution packages except for the unsolicited interrupt software from Fort Lewis.

(1) Software changes. TAB,TRAC-FLVN resolved differences between the three versions (each over 100 modules). We tried the unsolicited tablet interrupts from the Fort Lewis package with our Ramtek Marquis driver, but they did not work. We removed the hard-wired constants that tailored the various Kellner packages to specific Ramtek hardware, including resolution, DAC size, video board selectors, tablet and cursor selectors, MCP and memory-group selectors, and bit-plane masks. TAB,TRAC-FLVN replaced those constants with variables, some set by the FLIK station software and others read back from the Ramtek. See TRAC-F-TM-0687, <u>Fort Leavenworth Improved Kellner Station Package (FLIK-STN)</u> documentation. We merged similar Kellner modules into "super" modules having several entry points per module.

(2) Software additions. TAB,TRAC-FLVN wrote and documented the FLIK station package. That package maintained a unique data structure describing the Ramtek hardware at each user site. We designed and coded a new KOVRDEF color control module with what we feel to be smoother more generic logic. We wrote the first TAB,TRAC-FLVN version of OMNI, the Ramtek graphics training tool.

(3) Capabilities. Version 1.0 provided a single FLIK library to link to. This graphics library resolved for most applications the low/high resolution problem. Version 1.0 also adjusted user's Ramtek integer coordinates to fit the refresh memory size on the Ramtek in use.

b. FLIK Version 2.0. This was a major upgrade to help the VIP package work with the multiple-MCP stations at TRAC-FLVN. This version incorporated the following software: FLIK Version 1.0 and the first version of the FLIK station software. See

TRAC-F-TM-0687, the FLIK-STN documentation.

(1) Software changes. TAB,TRAC-FLVN went to a faster version of the FLIK-STN software that worked with an unformatted station file. This is still the current version of FLIK-STN. We also made fixes to KOVRDEF color control module.

(2) Software additions. We added the smarts in the imaging system to relate host pixel-bits with Ramtek bit-planes. FLIK could then keep a default mapping between bits in a host pixel and the Ramtek bit-planes targeted during a write-image. We also added FLIK modules that let applications define and use other such relationships. For more information, see paragraph 2g and appendix B. We also added a second write-image module that allowed double buffering of image data and solved the simultaneous image problem. For more information, see paragraph 5b and appendix E. With double buffering, an application could send pre-made pictures from disk to Ramtek twice as fast as before. We added the logic to KOVRDEF to automatically switch to a different MCP on the same station (if available) while allocating Ramtek planes to the user-specified color levels. We added the display list software and symbolic-names tables to let users reference Ramtek memory locations with symbolic names, providing relocatable display lists code. For more information, see appendix G. We also added a copy-image-and-multiply module so VIP or any other user could display low-resolution image files on a full high-resolution Ramtek screen without hardware zoom.

(3) Capabilities. FLIK Version 2.0 capabilities were the same as version 1.0 but were now more generic with respect to the number of MCPs per station and the bit-planes available. Version 2.0 made it much easier to transfer images. Version 2.0 initialized much faster due to new FLIK-STN software. See TRAC-F-TM-0687 documentation. When two processes simultaneously displayed images on different stations using the same Ramtek controller, if those processes used the new KWIMAGEEFN module, there was no possibility for the controller to hang. VIP could display low-resolution maps on a high-resolution Ramtek full screen. So, with Version 2.0, VIP users could keep only one set of picture files, and these files would each be around 642 blocks in size (instead of 2,566 blocks). Within a given amount of disk space, the VIP user could fit four times as many pictures.

c. FLIK Version 2.1. This version contained minor fixes to modules, and incorporated the following software: FLIK Version 2.0, and the newer, faster version of the FLIK-STN station software.

(1) Software changes. TAB,TRAC-FLVN modified KOVRDEF to properly handle overlay levels requiring up to and including 16 bit-planes. There are 16 planes available to each station on the

modified TRAC-FLVN 9460s. We also modified KTABLET to return integers instead of reals for the Ramtek coordinates of the cursor. We modified some modules to partially implement the new FLIK debug on/off control capability.

(2) Software additions. We added FLIK debug control modules DBG_FLIK and DBG_ONOFF.

(3) Capabilities. FLIK version 2.1 had the same capabilities as before. Also, the user could call DBG_ONOFF (string, logical) to control debug in the named module for on or off. This partial implementation only works with a few FLIK modules.

d. FLIK Version 3.0. This version includes a metafile input/output package and minor adjustments to work with VIC version 1.2. This version incorporates the following software: version 2.1 and the newer faster station software.

(1) Software changes. TAB,TRAC-FLVN changed 21 graphics output modules to write metafiles using the metafile package added to FLIK. We made new versions of the color-table modules KLOADVLT and KREADVLT to convert between user's expected and actual Ramtek color formats.

(2) Software additions. We added a metafile package to make new metafiles, write to them, and redisplay the contents of existing metafiles using a new driver that calls pre-existing FLIK graphics output modules. We added a command language symbol called FLIK_METAS and a corresponding foreign CLI command called FLIK_META. These let you redisplay a set of metafiles or a single metafile. See appendix F for additional information. We added modules KLAM and KRAM (old versions of modules KLOADVLT and KREADVLT that we upgraded).

(3) Capabilities. FLIK version 3.0 capabilities are the same as previous versions. Also, 21 FLIK modules now use the metafile package. Version 3.0 supports the requirement TRAC-WSMR placed on the Kellner graphics package used with VIC 1.2. The requirement was that, during normal VIC 1.2 graphics output, the Kellner package would save on a file the arguments passed from VIC 1.2 into 21 Kellner modules. After a VIC run, the user could run an off-line program to replay a picture from the file of saved arguments. TAB,TRAC-FLVN upgraded FLIK to provide a similar function. The corresponding 21 modules in FLIK save their input arguments on a metafile (see appendix F). Now, we can link FLIK to VIC 1.2 and generate metafiles, then use the command

```
$ FLIK_METAS  <filespec>
```

to redisplay those metafiles. Now, KLOADVLT and KREADVLT are smarter. These lower-level modules load and read back the VLT contents. Previously, these contents had the format of the DAC size of the current Ramtek station. However, with this upgrade, these two modules do a conversion between user-expected and actual Ramtek DAC sizes. That way, the application is free to manipulate VLT contents according to the expected DAC size (third argument in KRMINIT initialization call).

9. Summary. This memorandum discussed problems solved by FLIK, showed how to convert applications from Kellner packages to FLIK, examined the hardware-dependent considerations, and presented an ordered list of the development of FLIK versions thus far. We presented the basic purpose of FLIK: how it will make existing and new DOD graphics products that had used the Kellner graphics interface package transportable and usable at a large number of installations. Existing applications require very few changes to convert to the FLIK system. This transportability significantly increases the value of those products and, in some cases, may help DOD avoid costly duplication of effort. Please direct any comments or suggested improvements to Mr. Tim Daniels, Technology Applications Branch, TRAC-FLVN, AV 552-4261.

APPENDIX A

TABLET POLLING APPLICATION

The Kellner graphics tablet read services would always return to the user immediately: they would not wait for a puck button to he pushed. Therefore, applications using Kellner graphics that were designed to wait and effectively do nothing until a Ramtek puck button push occurred would execute a loop containing only a call to the Kellner tablet read service. The loop would continue until a puck button was pushed. However, the applications that actually required polling still used the Kellner tablet read calls in a similar polling loop, but that loop also contained operations in addition to the tablet read call. These additional operations were usually display updates of some kind, based on the current puck location. The following excerpt from VIC playback illustrates such a loop and how to change that software to use FLIK graphics.

```
CC    FROM VIC PLAYBACK SOFTWARE, WITH COMMENTS ADDED AND
CC    CODE REMOVED.
 10    CONTINUE
CC                 GET CURRENT CURSOR X/Y, CONVERT TO REAL-VIRTUAL
CC                 COORDINATES, AND DISPLAY ON VT220.
      CALL KRCURS(ICURSOR,IXCUR,IYCUR,ISTAT,ITRACK,IENTER)
      CALL KTRANRTV(IXCUR,IYCUR,XPOS,YPOS)
      CALL CURSOR(11,1)

C  CALL UTILITY ROUTINE WITH PREVIOUSLY CALCULATED X,Y OFFSETS
C  (XOFFST,YOFFST) ADDED TO XPOS,YPOS. RETURNS UTM
CC CONVERT MC TO UTM, AND DISPLAY ON VT220.
CC
CC          FOLLOWING LINE CONTAINS AN ERRONEOUS
CC          ASSUMPTION EXPLAINED BELOW.
      IBUTN=IENTER

C******************************************************
C    *AN ENHANCED ROUTINE IN FLIK, KTABLET, ALLOWS THE USER TO DO
C    *POLLING AND NON-POLLING TABLET READS.
C    * ACTUALLY, THE ABOVE KRCURS AND KTRANRTV CALLS ARE
C    * UNNECESSARY SINCE KTABLET RETURNS BOTH THE
C    * SCREEN AND VIRTUAL COORDINATES OF THE CURSOR.
C    * TO CLEAN UP THIS ROUTINE SHOULD MOVE THE
C    * KTABLET CALL TO WHERE THE KRCURS CALL
C    * IS AND DELETE IT AND THE
C    * KTRANRTV CALL ALTOGETHER.

CCCCCCCALL KRTABBUT(ICURSOR,IDUMX,IDUMY,IBUTN)
      CALL KTABLET (0, IBUTN, IXCUR, IYCUR, XPOS, YPOS)
C***********************************************************
CC                     IF PUCK BUTTON NOT PRESSED YET, THEN
CC                     LOOP BACK AND DISPLAY COORDINATES AGAIN.
      IF(IBUTN.EQ.0) GO TO 10
      CALL KCTEXT(IXCUR-7,IYCUR+8,IHGHT1,'O',1)
```

APPENDIX B

HOST MODE IMAGE SETTING

The Kellner packages each worked with a specific type of Ramtek having a set number of bit-planes. The application built its images for that set number of planes. When using FLIK, one single graphics package allows for several possibilities of bit-planes and image formats. You must ensure that applications correctly specify the image format in the first arguments to KWIMAGE and KRIMAGE. You must also ensure that users call these modules and not ones specially added to some Kellner package. The following example, an excerpt from a VIP/VISTA routine, shows such a case.

```
CC       VIP-SUBROUTINE VISTA_PERS_PERSPECTIVE_MAP
CC       (after image ready to transfer)
CC    .         .           .
CC    .         .           . CODE THAT PREPARES THE VIEW IN
CC    .         .           . HOST APPLICATION VIRTUAL MEMORY
CC    .         .           .
C  DRAW THE PERSPECTIVE VIEW IN THE WINDOW
      DO 600 IY=0,NYSPL-1,NROWS
          IX1 = NXOFSPL
          IY1 = NYOFSPL + IY
          IX2 = NXOFSPL + NXSPL - 1
          IY2 = NYOFSPL + IY + NROWS - 1
          IF (IY2.GE.NYOFSPL+NYSPL) IY2 = NYOFSPL+NYSPL-1
          IF( ICONFIG .EQ. 1 ) THEN
              CALL KWIMAGE(1,IX1,IY1,IX2,IY2,PIXELBUF1(IPNDX))
          ELSE
CC                                UNUSED
CC                                ARGUMENT
CC                                  V
CCCCCCCCCCCCCCCCALL KWIMAGE_8600_V(1,IX1,IY1,IX2,IY2,
     *             PIXELBUF2(IPNDX))
              CALL KWIMAGE(1,IX1,IY1,IX2,IY2,PIXELBUF2(IPNDX))
          ENDIF
```

APPENDIX C

GET AVAILABLE BIT-PLANES

This example shows how to determine the number of bit-planes available on the current Ramtek station attached to.

```
CC                 YOU NEED A SMALL ARRAY FOR INFORMATION
CC                 FED BACK BY THE RAMTEK STATION SOFTWARE.
      PARAMETER LEN_LIST = 32
      INTEGER * 4 PIX_LIST (4, LEN_LIST)
CC                DO THE NORMAL INITIALIZATION, IF NOT ALREADY
CC                DONE PREVIOUSLY IN THE APPLICATION.
      CALL KRMINIT (640, 512, 4, IMON)
CC                NOW GET THE NO. OF USABLE BIT-PLANES ON THIS
CC                STATION. ARRAY TELLS FOR EACH BIT-PLANE
CC                THE MCP, GROUP, AND ACTUAL PLANE NUMBER.
      CALL RMSTN_PIXBITS (LEN_LIST, NBITS, PIX_LIST, IERR)
```

APPENDIX D

IMAGE SIZE IN APPLICATIONS

This example shows how to make applications sensitive to the image size and how an application can respond to varying numbers of available bit-planes. This slightly rearranged code comes primarily from VIP modules ASSIGNGRAPH SAVEPIC, and PWRITER. Note the calls to KQWIN and RMSTN_PIXBITS to tell the application the Ramtek resolution and number of available bit-planes. Note also the calls to KPIXPUSH, KPIXHDFAULT, and KPIXRMDFAULT that determine the relation between host pixel bits and Ramtek bit-planes used by FLIK during image transfers.

```
CC              YOU MUST MAKE A SMALL ARRAY TO CONTAIN
CC              INFO RETURNED BY THE RAMTEK STATION MODULE.
      PARAMETER LEN_LIST = 32
      INTEGER * 4 PIX_LIST (4, LEN_LIST)
CC             SPACE FOR INSTRUCTION FIELD AND IMAGE-DATA FIELD.
CC             INSTRUCTION FIELD IS 11 WORDS.
CC             IN THE DATA FIELD, USE EITHER
CC             ONE BYTE OR ONE WORD PER PIXEL.
CC             FOR LOW-RES, MAKE SPACE FOR 64 LINES, 640
CC             PIXELS PER LINE.  FOR HI-RES, MAKE SPACE FOR
CC             32 LINES, 1280 PIXELS PER LINE.
      COMMON /PIC005/ PICBUFF1, PICBUFF2
      BYTE         PICBUFF1 ( 40982 , 2 )
                            ! 22 BYTES OF INSTR + 40960 OF DATA
      INTEGER * 2  PICBUFF2 ( 40971 , 2 )
                            ! 11 WORDS OF INSTR + 40960 OF DATA
      EQUIVALENCE (PICBUFFF1 (1, 1), PICBUFF2 (1, 1)
CC
      LOGICAL HIRES, HALF_WORD  ! RAMTEK CHARACTERISTICS
      INTEGER * 2 QWIN (4)      ! FORMAT WINDOW SIZE
      CHARACTER*132 PICNAME     ! DISK FILE NAME
CC              INITIALIZE IF NOT DONE SO PREVIOUSLY.
CC              THE LOW-RES 4-BIT DAC IS MERELY AN EXAMPLE.
      CALL KRMINIT (640, 512, 4, IMON)
CC           GET THE RAMTEK STATION RESOLUTION AND SET FLAG.
      CALL KQWIN (QWIN)
      HIRES = QWIN (3) .GT. 640
CC            GET THE NUMBER OF BIT-PLANES AVAILABLE.
      CALL RMSTN_PIXBITS (LEN_LIST, NBITS, PIX_LIST, IERR)
CC              SET THE "ACTUAL" IMAGE-RECTANGLE TO COVER
CC              THE ENTIRE REFRESH MEMORY (FULL-SCREEN)
      XRESMN = QWIN (1)
      YRESMN = QWIN (2)
      XRESMX = QWIN (3)
      YRESMX = QWIN (4)
CC    .                .
CC    .                .IF NOT DONE PREVIOUSLY, THE
CC    .                .SOFTWARE TO DRAW PICTURE GOES HERE.
CC    .                .
CC
```

```
CC                SAVE THE IMAGE ON DISK. MAKE DISK RECORDS
CC                EACH CONTAINING EITHER 64 LOW-RES OR
CC                32 HIGH-RES HORIZONTAL LINES, FULL-WIDTH.
CC
CC           SET XLL,YLL,XUR,YUR FOR A FULL SCREEN
      IXPMN = XRESMN
      IYPMN = YRESMN
      IXPMX = XRESMX
      IYPMX = YRESMX
CC               SET A PICTURE FILE NAME FOR THIS EXAMPLE.
      PICNAME = 'FLIK_EXAMPLE'
CC               IF A CERTAIN FILE NAME AND IF .GT. 8 BIT-PLANES
CC               ON THIS STATION, SWITCH TO HALFWORD IMAGES.
      INDX = INDEX (PICNAME, 'VISTA_WINDOW' )
      IF (INDX .NE. 0 .AND. NBITS .EQ. 10 ) THEN
          HALF_WORD = .TRUE.
          IHMODE = 0
      ELSE
          HALF_WORD = .FALSE.
          IHMODE = 1
      ENDIF
CC              SAVE THE CURRENT IMAGE TRANSFER RELATION BETWEEN
CC              HOST PIXEL-BITS AND RAMTEK BIT-PLANES.
      CALL KPIXPUSH (IERR)
CC              RESET TO DEFAULT RELATION BETWEEN HOST BITS
CC              AND RAMTEK BIT-PLANES.
      CALL KPIXHDFAULT
      CALL KPIXRMDFAULT
CC              OPEN THE HOST DISK FILE TO SAVE THE PICTURE.
      OPEN ( UNIT = 12, FILE            =   PICNAME     ,
     *                    ACCESS        =  'SEQUENTIAL'  ,
     *                    FORM          =  'UNFORMATTED' ,
     *                    ORGANIZATION  =  'SEQUENTIAL'  ,
     *                    STATUS        =  'UNKNOWN'     )
C                           SET NUMBER OF X SCAN LINES TO PROCESS
C                           AT ONE TIME BASED ON THE RESOLUTION.
      IF ( HIRES) THEN
          NPIX = 32
      ELSE
          NPIX = 64
      ENDIF
CC                    IF WORKING WITH HALFWORDS,
CC                    CUT THE NUMBER OF LINES IN HALF.
      IF (HALFWORD) THEN
          NPIX = NPIX / 2
      ENDIF
C                                CALCULATE X AND Y RANGE
      NXPIX = IXPMX - IXPMN + 1
      NYPIX = IYPMX - IYPMN + 1
      NXYPIX = NYPIX - 1
      IBUFSIZE =  NXPIX * NPIX
CC                    WRITE A HEADER RECORD CONTAINING
CC                    THE PIXEL FORMAT, "ACTUAL" IMAGE
CC                    WIDTH HEIGHT, LINES-PER-RECORD, AND
CC                     LOWER-LEFT AND UPPER-RIGHT CORNERS,
CC                     AND THE RAMTEK RESOLUTION.
      INFO = IDIR
      IF (HALF_WORD) INFO = IOR (INFO, '10'X)
      WRITE(12) INFO, NXPIX, NYPIX, NPIX,
     *            IXPMN, IYPMN, IXPMX, IYPMX, QWIN
CC                     LOOP ON HORIZONTAL IMAGE SECTIONS.
      DO 100 I = 0, NXYPIX, NPIX
          IXLL = IXPMN
          IYLL = IYPMN + I
          IXUR = IXPMX
          IYUR = IYPMN + I + NPIX - 1
          IF ( IYUR .GT. IYPMX ) IYUR = IYPMX
CC                   T. DANIELS: 15-MAR-1988: ADDED IF-CHECK
CC                   ON HALF_WORD FORMAT
CC                   IF WORKING WITH A PICTURE REQUIRING
CC                   MORE THAN EIGHT BIT-PLANES, THEN
          IF (HALF_WORD) THEN
              CALL KRIMAGE ( IHMODE, IXLL, IYLL, IXUR, IYUR,
     *               PICBUFF2 (12, 1) )
          ELSE
              CALL KRIMAGE ( IHMODE, IXLL, IYLL, IXUR, IYUR,
     *               PICBUFF1 (23, 1) )
          ENDIF
C                         CALCULATE NUMBER OF WORDS TO WRITE OUT
          NWDS = (( IXUR - IXLL ) + 1 ) * (( IYUR - IYLL) + 1 )
   NWORDS = MIN ( NWDS , IBUFSIZE )
CC                   IF WORKING WITH A PICTURE REQUIRING
CC                   MORE THAN EIGHT BIT-PLANES, THEN
          IF (HALF_WORD) THEN
CC                     DOUBLE THE BYTE COUNT.
              NWORDS = NWORDS * 2
              CALL PWRITER ( PICBUFF2 (12, 1) , NWORDS )
          ELSE
              CALL PWRITER ( PICBUFF1 (23, 1) , NWORDS )
          ENDIF
  100 CONTINUE
      CLOSE ( UNIT = 12 )
CC                RESTORE THE PREVIOUS IMAGE TRANSFER
CC                ENVIRONMENT.
      CALL KPIXPOP (IERR)
      RETURN
      END
      SUBROUTINE PWRITER ( IA, NWDS )
      BYTE IA(NWDS)
      WRITE ( 12 ) IA
      RETURN
      END
```

APPENDIX E

SIMULTANEOUS IMAGE TRANSFER PROBLEM

E-1 General. Below we describe the cause and solution for the problems that occur when attempting with separate processes to simultaneously redraw saved images from disk onto different Ramtek stations driven by the same Ramtek controller.

E-2. Cause. To explain the root cause of the multistation simultaneous image transfer problem, we present the following topics.

o Multistation Ramteks and instruction handling.

o VAX/VMS I/O services.

o VAX/VMS I/O to multistation Ramteks.

o Marquis driver station-switching.

o Ramtek instruction formats.

o Kellner and FLIK Ramtek-instruction building.

o Instruction transfers and splitting.

o Split instructions, interlacing, and multiple stations.

o Confusion from split instruction in the driver and controller.

a. Multistation Ramteks and instruction handling. A graphics station partially consists of a single Ramtek monitor and one or more input devices (such as tablet, cursor controller, and/or Ramtek keyboard). These are all peripherals for a single user. The remainder of the station is inside the Ramtek controller, on several circuit boards dedicated to that station, and on one circuit board shared by all stations on that controller. This shared board is called the display controller and is the focus of this discussion. From a programmer's perspective, the display controller board's main task is to handle instructions sent from the host computer. It parses and translates those instructions, then feeds them to some station-specific circuit board that carries out those instructions.

b. VAX/VMS I/O services. Kellner and FLIK both transfer instructions to the Ramtek, where the display controller operates on them as described above. To do this transfer, Kellner or FLIK both must use system services that submit a request for input/output operations. Such a request is called a QIO, and with a VAX/VMS I/O channel number, refers to I/O operations on a specified peripheral device. A single QIO can only transfer one or more contiguous words of physical memory. (The term "contiguous" is one key to understanding the current problem.) When a process attaches to one of several stations on the same Ramtek controller, VAX/VMS gives it a unique I/O channel number to use in QIO operations to that station. Even with these unique channel numbers, VAX/VMS effectively puts onto a single I/O queue all QIOs to all stations on that controller. Separate channels for separate stations do not make for separate sequences of QIOs.

c. VAX/VMS I/O to multistation Ramteks. When users simultaneously run two processes attached to different stations on the same Ramtek controller, the QIO requests from within those processes intermingle on the same system I/O queue. Thus, this intermix of QIOs in VAX/VMS interleaves the previously separate streams of Ramtek instructions that had each been intended for a distinct Ramtek station. This intermix of QIOs and Ramtek instructions is another key to understanding the current problem.

d. Marquis driver station-switching. The Marquis driver, written in Macro Assembler and loaded with the VAX/VMS system, carries out the I/O requests passed to it by VMS from the system I/O queue. There is some attempt by the Marquis driver to resolve this problem of the intermix of instruction streams tagged for separate Ramtek stations. By using the I/O channel included in the QIO request as a station-tag, the driver determines the target station for each QIO it handles. The driver processes a single mixed stream of QIOs, one QIO at a time. The driver targets this QIO stream at a "current" Ramtek station. When the driver detects that the current QIO is meant for a Ramtek station different from the "current target station", the driver switches "current target station" to that new Ramtek station. Part of that switching is to send instructions to the Ramtek itself to change to the context and graphics environment of that new station.

(1) The driver "current target station" concept has two purposes: it relieves the host graphics package or its user from keeping the correct Ramtek graphics context for the current Ramtek station, and it effectively separates the interleaved Ramtek instruction streams back into the original sets tagged by I/O channel number for each station.

(2) There is an assumption fundamental to the driver's reliance on the I/O channel number in each QIO as a station-tag when sorting the merged Ramtek instruction streams. The assumption is that a single Ramtek instruction will never be split into more than one QIO. (This assumption is another major piece of the problem).

e. Ramtek instruction formats. There are two general forms of Ramtek instructions--single- and double-field. The single-field instruction usually has an operation code, followed by a few parameters, and is usually quite short. The double-field instruction has an instruction field with an operation code and parameters followed by a variable-length data field. The write-image is a double-field instruction with a small instruction field and a data field that can sometimes be very large.

f. Kellner and FLIK Ramtek-instruction building. The user application calls upper-level graphics modules in either Kellner or FLIK. In response to those calls, the graphics package builds the Ramtek instructions necessary to satisfy the user application's graphics requirements, and when complete, transfers those Ramtek instructions to the Ramtek with QIO requests. Kellner and FLIK both build single-field instructions one way and double-field instructions another way. When building the single-field instructions, they locate all words of each such instruction in contiguous memory internal to the graphics package. However, when building double-field instructions, both Kellner and FLIK locate the instruction and data fields of each such instruction in two separate noncontiguous parts of host memory. However, each of the two parts is contiguous. Usually the instruction field is inside Kellner or FLIK and the data-field is with the application. The instructions are now ready for transfer to the Ramtek.

g. Instruction transfers and splitting. There are two instruction-transfer "modes" in which Kellner and FLIK can operate--nonbuffered (slow) and buffered (fast). In their slow nonbuffered mode, both Kellner and FLIK transfer each instruction immediately after it is built. They send each single-field instruction with a single QIO request, and each double-field instruction with two QIO requests.

(1) In the buffered mode, the treatment of Ramtek instructions just after building them depends on the instruction length. The normal case is for shorter instructions while the exception case is for very long instructions. In the normal case, both Kellner and FLIK copy each of the single-field instructions and each of the shorter double-field instructions into an internal graphics-output buffer immediately after building them. There, those instructions await transfer to the Ramtek. Later, a single QIO will transfer all instructions saved in the graphics output buffer, after which the buffer is cleared. That transfer and clear operation occurs for three reasons: (1) the buffer fills, (2) the user asks to dump the buffer or read back cursor, keyboard or tablet information, and (3) Kellner or FLIK processes an instruction that is an exception case. The exception case occurs just after building the instruction (first) field of a very long double-field instruction having a large data (second) field. In such a situation, Kellner or FLIK copies only the instruction (first) field into the graphics output buffer, does a normal transfer and clear operation as described above, then sends the entire large data field of this current double-field instruction with a single separate QIO.

(2) There are two situations in which a single instruction will be split into multiple QIOs. They are: when sending any double-field instructions during nonbuffered transfer mode, and when sending double-field instructions having a large data-field during buffered mode.

h. Split instructions, interlacing, and multiple stations. When running just one process that transfers instructions to the Ramtek, there is only one contributor to the system queue of QIOs for that Ramtek controller. In addition, all of those QIOs are tagged for the same station on that Ramtek controller. As a result, the potential situation never arises that VAX/VMS would interlace the QIOs for several stations on that Ramtek controller. Even in the cases described above, when Kellner or FLIK produce pairs of QIOs for some single instructions, those pairs always stay together in the sequence of QIOs passed by VMS to the Marquis driver. But, it is a completely different situation when running two processes that simultaneously transfer Ramtek instructions to different stations on the same Ramtek controller. The QIOs to transfer instructions to those several different stations will most assuredly interlace on the system queue. Interlacing of QIOs results in interwoven blocks of Ramtek instructions to be passed by the driver, some for one station, and some for another. If any of those QIOs should happen to include QIO pairs generated by Kellner or FLIK to transfer some individual instruction, then those pairs may well be separated from each other in the VAX/VMS queue as a result of the interlace of QIOs.

i. Confusion from split instructions in the driver and controller. This happens quite frequently to lengthy write-image instructions. Under such conditions of two processes redrawing images at the same time, VAX/VMS often sends to the Marquis driver these QIO pairs interlaced in the sequence shown below.

- o The instruction-field for the first station's write image.

- o The instruction-field for the other station's write image.

- o The data-field for the first station's write image.

- o The data-field for the second station's write image.

While testing the channel numbers on each QIO, the driver will inject its usual change-station commands and add to the confusion. Consider what happens to this sequence on the Ramtek side. The display controller firmware first parses the instruction field of the first write-image. That field specifies that a certain number of words IMMEDIATELY CONTIGUOUSLY FOLLOWING the instruction field will contain the data field of that same write-image instruction. So, the display controller treats them as such, even though due to the intermix of QIOs and the driver's station-switching functions, the first few words following are the switch-context instructions and the instruction field of the second write-image instruction, intended for the other Ramtek station. Following that, the display controller finds most of the data field of the first write-image and treats it as such. The display controller then treats the last few words of the data field of the first write image instruction as a Ramtek instruction, which usually blows it up since it is parsing data, not instructions.

j. Summary of the problem. To sum up thus far, this problem occurs when we run two simultaneous processes each assigned one station on a multistation controller, sending QIOs over separate I/O channels but through a single system queue wherein the requests mix. This results in an interleaved sequence of Ramtek instructions transferred, some for one station, and some for the other. The Marquis driver that acts for both stations handles each QIO individually and switches the Ramtek into the proper station context indicated by the I/O channel for that QIO. The driver assumes that Ramtek instructions never overlap between QIOs. The write-image instruction usually has a very large data-field. Therefore, Kellner and FLIK send each of those instructions using a QIO pair. Often these pairs get split by the interlace effect of the I/O queue, and due to the driver's assumption that Ramtek instructions are never split in such a manner, the driver will fail to undo the mix of the instructions tagged for each station, and will inject in the wrong places the switch context instructions. This results in a garbled Ramtek instruction and the display controller fails to parse it.

E-3 Solution. To solve the simultaneous imaging problem, we tackled the software that we could change, namely FLIK. Since we could not prevent the interlacing of QIOs or alter the system queue logic, we needed the ability to send write-image instructions with one QIO. To do that, the entire instruction needed to be contiguous in physical memory. We had two design options: to greatly increase the size of the graphics output buffer to contain large write-image instructions when in buffered mode (around 65,545 words), or to provide a new FLIK module that builds the write-image instruction field in user space just prior to the data field. We have done the latter.

a. The new FLIK module is KWIMAGEEFN. This module requires that you use a slightly larger image buffer than before, with the first 11 halfwords set aside for the Ramtek instruction. You must be sure to call KWAIT after calling KWIMAGEEFN. KWIMAGEEFN was originally used in double buffering operations, but you can use it in single buffer operations as well.

b. Once called, KWIMAGEEFN puts the instruction-field of the write-image instruction into those additional 11 halfwords and then issues one single QIO call to send the contiguous instruction and data fields of the entire write-image instruction. We left the original KWIMAGE alone, for compatibility with existing user applications.

E-4. Sample solution. The following example comes from VIP where TAB at TRAC-FLVN first tested the double buffering in FLIK. This example shows the following things: how to provide the 11 extra instruction words (required for successful simultaneous-process image transfers) at the front of the application image buffer, how to do double buffering, and how to make image-handling sensitive to the Ramtek resolution and to the number of bit-planes available on a station.

```
CC              YOU MUST MAKE A SMALL ARRAY TO CONTAIN
CC              INFO RETURNED BY THE RAMTEK STATION MODULE.
      PARAMETER LEN_LIST = 32
      INTEGER * 4 PIX_LIST (4, LEN_LIST)
CC             SPACE FOR INSTRUCTION FIELD AND IMAGE-DATA FIELD.
CC             INSTRUCTION FIELD IS 11 WORDS.
CC             IN THE DATA FIELD, USE EITHER
CC             ONE BYTE OR ONE WORD PER PIXEL.
CC             FOR LOW-RES, MAKE SPACE FOR 64 LINES, 640
CC             PIXELS PER LINE.  FOR HI-RES, MAKE SPACE FOR
CC             32 LILNES, 1280 PIXELS PER LINE.
      COMMON /PIC005/ PICBUFF1, PICBUFF2
      BYTE         PICBUFF1 ( 40982 , 2 )
                             ! 22 BYTES OF INSTR + 40960 OF DATA
      INTEGER * 2  PICBUFF2 ( 40971 , 2 )
                             ! 11 WORDS OF INSTR + 40960 OF DATA
      EQUIVALENCE (PICBUFFF1 (1, 1), PICBUFF2 (1, 1)
CC
      LOGICAL HIRES, HALF_WORD  ! RAMTEK CHARACTERISTICS
      INTEGER * 2 QWIN (4)      ! FORMAT WINDOW SIZE
      LOGICAL START_PIC, END_PIC
      LOGICAL EXIST
CC
      INTEGER * 2 WMSK
      INTEGER * 2 IWIN (4)
CC
      CHARACTER*132 PICNAME     ! DISK FILE NAME
CC
      PICNAME = 'FLIK_EXAMPLE'
CC              INITIALIZE IF NOT DONE SO PREVIOUSLY.
CC              THE LOW-RES 4-BIT DAC IS MERELY AN EXAMPLE.
      CALL KRMINIT (640, 512, 4, IMON)
CC           GET THE RAMTEK STATION RESOLUTION AND SET FLAG.
      CALL KQWIN (QWIN)
      HIRES = QWIN (3) .GT. 640
CC            GET THE NUMBER OF BIT-PLANES AVAILABLE.
      CALL RMSTN_PIXBITS (LEN_LIST, NBITS, PIX_LIST, IERR)
CC              SET THE "ACTUAL" IMAGE-RECTANGLE TO COVER
CC              THE ENTIRE REFRESH MEMORY (FULL-SCREEN)
      XRESMN = QWIN (1)
      YRESMN = QWIN (2)
      XRESMX = QWIN (3)
      YRESMX = QWIN (4)
```

```
CC           SET XLL,YLL,XUR,YUR FOR A FULL SCREEN
CC
CC                SAVE THE CURRENT IMAGE TRANSFER ENVIRONMEMNT
CC                AND SET TO DEFAULT RELATION BETWEEN
CC                HOST PIXEL-BITS AND RAMTEK BIT-PLANES.
      CALL KPIXPUSH (IERR)
      CALL KPIXHDFAULT
      CALL KPIXRMDFAULT
CC                CHECK TO SEE IF FILE EXISTS
      INQUIRE ( FILE = PICNAME, DEFAULTFILE = 'PICFILE:',
     1           EXIST = EXIST )
      IF ( .NOT. EXIST ) THEN
          PRINT*,' PICTURE FILE DOES NOT EXIST ',PICFLNAME
          PICNAME = '$NOT$EXIST$'
          RETURN
      END IF
      OPEN ( UNIT = 12, FILE         =   PICFLNAME     ,
     *                  DEFAULTFILE   =  'PICFILE:',
     *                  ACCESS        =  'SEQUENTIAL'  ,
     *                  FORM          =  'UNFORMATTED' ,
     *                  READONLY,
     *                  ORGANIZATION  =  'SEQUENTIAL'  ,
     *                  STATUS        =  'OLD'         )
CC          READ IN HEADER RECORD FOR PICTURE FILE
      READ ( 12 ) INFO,NXPIX, NYPIX, NPIX,
     *              IXPMN, IYPMN, IXPMX,IYPMX, QWIN
CC                  CHECK THE HALF-WORD BIT.
      IF (IAND (INFO, '10'X) .NE. 0) THEN
          HALF_WORD = .TRUE.
          IHMODE = 0
      ELSE
          HALF_WORD = .FALSE.
          IHMODE = 1
      ENDIF
CC            SET NUMBER OF POINTS
      NXYPIX = NYPIX - 1
      IBUFSIZE = NXPIX * NPIX
CC           USE TWO PIXEL BUFFERS.
CC           FIRST READ IS INTO BUFFER ONE.
      IN_PIC = 1
      IO_PIC = 2
CC            THIS LOOP WILL READ IN PICTURE IN SLICES.
      DO 100 INDX = 0, NXYPIX, NPIX
CC            SET LOOP-CONDITION FLAGS
          START_PIC = (INDX        .EQ. 0)
          END_PIC   = (INDX + NPIX .GT. NXYPIX)

CC                  IF THIS IS FIRST PASS THROUGH LOOP, THEN
          IF (START_PIC) THEN
C                             IF DIRECTION IS LEFT TO RIGHT THEN
              I = INDX
       IXLL = IXPMN
       IYLL = IYPMN + I
       IXUR = IXPMX
```

```
            IYUR = IYPMN + I + NPIX - 1
            IF ( IYUR .GT. IYPMX ) IYUR = IYPMX
C                                   CALCULATE NUMBER OF WORDS TO READ IN
                NWDS = (( IXUR - IXLL ) + 1 ) * (( IYUR - IYLL ) + 1 )
                NWORDS = MIN ( NWDS , IBUFSIZE )
                   IF (HALF_WORD) THEN
CC                       DOUBLE THE BYTE COUNT.
                       NWORDS = NWORDS * 2
CC                           READ THE FIRST BUFFER
                CALL PREADER ( PICBUFF2 (12, IN_PIC), NWORDS )
                   ELSE
CC                           READ THE FIRST BUFFER
                CALL PREADER ( PICBUFF1 (23, IN_PIC), NWORDS )
                   ENDIF
               ENDIF
CC                    SWAP BUFFERS
               ITEMP = IO_PIC
               IO_PIC = IN_PIC
               IN_PIC = ITEMP
CC                      WRITE CURRENT BUFFER
               IF (HALF_WORD) THEN
            CALL KWIMAGEEFN ( IHMODE, IXLL, IYLL, IXUR, IYUR,
          *            PICBUFF2 (1, IO_PIC))
               ELSE
            CALL KWIMAGEEFN ( IHMODE, IXLL, IYLL, IXUR, IYUR,
          *            PICBUFF1 (1, IO_PIC))
               ENDIF
CC              IF NOT DONE READING, THEN
               IF (.NOT. END_PIC) THEN
C                                    IF DIRECTION IS LEFT TO RIGHT THEN
                  I = INDX + NPIX
            IXLL = IXPMN
            IYLL = IYPMN + I
            IXUR = IXPMX
            IYUR = IYPMN + I + NPIX - 1
            IF ( IYUR .GT. IYPMX ) IYUR = IYPMX
C                                   CALCULATE NUMBER OF WORDS TO READ IN
                NWDS = (( IXUR - IXLL ) + 1 ) * (( IYUR - IYLL ) + 1 )
                NWORDS = MIN ( NWDS , IBUFSIZE )
                   IF (HALF_WORD) THEN
                       NWORDS = NWORDS * 2
CC                    READ NEXT BUFFER
                CALL PREADER ( PICBUFF2 (12, IN_PIC), NWORDS )
                   ELSE
CC                    READ NEXT BUFFER
                CALL PREADER ( PICBUFF1 (23, IN_PIC), NWORDS )
                   ENDIF
               ENDIF
CC                    WAIT FOR WRITE TO FINISH
               CALL KWAIT
  100 CONTINUE
          CLOSE ( UNIT = 12 )
CC              RESTORE THE PREVIOUS IMAGE-TRANSFER
CC              ENVIRONMENT.
```

```
CALL KPIXPOP (IERR)
RETURN
END
SUBROUTINE PREADER ( IA, NWDS )
BYTE IA(NWDS)
READ ( 12 ) IA
RETURN
END
```

APPENDIX F

META-FILE CONTROL

F-1. Files creation. You control the creation of FLIK metafiles either from the terminal at run time or with a subroutine call from within the application. Control from the terminal occurs during initialization when you see this message·

```
 KRMINIT: Enter one of these, for specified effect
 Ramtek logical (like RMA0:)  (graphics only)
 Ramtek logical (like RMA0:M) (graphics and metafile)
 Ramtek logical (like RMA0:Y) (metafile only)
```

a. If you were to enter "RMA0:M" for example, FLIK would draw normal graphics plus create a metafile. Note that only the FLIK modules in the set of upgraded modules would put their information on that metafile. If you had entered "RMA0:Y" for example, you would see no graphics and could do no interactive Ramtek tablet inputs, but FLIK would still create that same metafile containing information from those same upgraded modules.

b. From within an application, you can control the metafile capability with a call to K_TOGGLE_FILE_WR module. It has one argument with valid inputs as follows:

```
 SUBROUTINE K_TOGGLE_FILE_WR(ION_OFF)

C ION_OFF = 1:  don't write to file but       draw Ramtek data.
C ION_OFF = 2:        write to file and       draw Ramtek data.
C ION_OFF = 3:        write to file but don't draw Ramtek data.
C ION_OFF = 4:        read     file and       draw Ramtek
data
```

c. Another way to produce metafiles is to assume that meta output is enabled. Then whenever you erase the screen with a KOERASE or KERASE, FLIK saves that command on the current metafile and closes it. FLIK then opens a new metafile and automatically writes the current user expectations and color overlay scheme onto that new file. The subsequent meta output goes to that new file until the application does another KOERASE or KERASE call. So, when running an application that includes these screen erases, you will create a chain of metafiles, where all but the last file ends in a screen erase. The metafile name is

[]METAnnn.DAT

where "nnn" is a three-digit number ranging from 000 to 999. While metafile creation is enabled, FLIK increases this number each time it closes one metafile and opens another.

F-2. Playback. Once FLIK produces a metafile for an application you can type it or look at it with the standard text editors. We include the following information which you can also see when you enter "HELP_FLIK FLIK_META DISP COMMAND" on the VAX/VMS system.

a. FLIK-META(S) command format. When replaying metafiles from the terminal, you have two choices. In both cases, the command format is the same, as follows:

```
$ FLIK_META <file-spec> <options>

$ FLIK_METAS <file-spec> <options>
```

(1) Options. The following switches are associated with these commands. They look like VMS DCL command switches, starting with a slash (/), but unlike DCL options you must spell them out completely.

(a) /TRACE - Use this switch to see a list of the FLIK modules played back, in order.

(b) /TOTAL - Use this switch to see the modules and total number of calls made for each file and for the entire set of files played back.

(c) /EXCLUDE=<module-list> - Use this switch to suppress the display of module(s) on the metafile.

(d) /ONLY=<module-list> - Use this switch to only display specified module(s) from the metafile. Note that initialization modules KRMINIT, KSELVO, and KOVRDEF are not suppressed by this ONLY switch.

(2) Module-list. This is a list of one or more FLIK modules. Used with the /EXCLUDE switch, the modules specified will be excluded from the FLIK-META replay. Used with the /ONLY switch, the modules specified will be the only ones displayed. The format for the <module-list> is either of the following.

```
o <module>

o ( <module> [,<module>]..  .)
```

b. Examples. The examples below show the use of the FLIK_META and FLIK_METAS commands, and explain their effect.

(1) Example one. The following command displays a single specified file.

```
$ FLIK_META [.TEST]META000.DAT
```

(2) Example two. The following command displays a single specified file with a count of the number of times each module is called. It excludes the display of rectangles through KRECT

module.

```
        $ FLIK_META [.TEST]META000.DAT /TOTAL /EXCLUDE=KRECT
```

(3) Example three. This command displays all files satisfying the file-spec shown.

```
        $ FLIK_METAS [.TEST]META*.DAT
```

(4) Example four. The following command displays all files satisfying the file-spec shown, with an ordered list of the modules called. Only the modules specified in parentheses get displayed from the metafile.

```
        $  FLIK_METAS [.TEST]META*.DAT /TRACE -
               /ONLY=(KOVRBUF,KOCOLOR,KRECT,KCTEXT)
```

APPENDIX G

DISPLAY LISTS

G-1. Display lists: general. Display lists are programs made of Ramtek instructions that you can load into random access memory on the Ramtek. You can call a display list to execute the instructions it contains. These calls can come either from the host or from another display list. Display lists can perform most normal graphics instructions. In addition, they can branch, load-store registers, do arithmetic and Boolean operations, call other display lists, and take data input from peripherals such as tablets or keyboards. For more complete information on FLIK's display list capabilities, enter the command HELP_DL in DCL on VAX/VMS.

G-2. FLIK display list software location. Most of the software for FLIK display list operations resides in libraries separate from regular FLIK. Use the logical OFLIKDL for the directory containing these libraries. There you will find text, object, and help libraries. In a subdirectory called [.REFERENCE_MANUAL] a printable document called FLIKDL.MEM describes these capabilities in greater detail. Any of the .MEM files at OFLIKDL or in subdirectories below that should be helpful. To pick up the new FLIK display list modules, insert one line in your application's link file, just above the line OKGL:FLIK/OPT, as follows:

```
$ LINK <main> -
.   .    .
.   .    .
.   .    .
OFLIKDL:A/LIB,-    ! new FLIK display list modules
OKGL:FLIK/OPT      ! standard FLIK graphics library
```

G-3. FLIK display list functions. FLIK now provides four basic display-list operations: define and compile, link, send, and call. These capabilities must operate in order, but not necessarily under control of the same application.

a. There are various ways in which one or more applications can carry out these four operations. For example, one run of a single application program could define and compile one or more display lists, link them together, and send the linked display lists to the Ramtek and later call them as needed. Alternatively, separate applications could each define and compile several different display lists. Later, the user could execute a stand-alone linker program to perform an off-line link of all those display-list files made earlier. Finally, a subsequent application program could send the linked display lists all together to the Ramtek and afterwards call them as needed.

b. The following modules under OFLIKDL control these four operations:

(1) DL_OPEN -- begins the definition and compile phase of a display list with a specified external symbolic name (see paragraph G-4 below). FLIK will eventually produce a file by that name. This module automatically provides an external entry point whose name is equal to the display list name. From here until a call to DL_CLOSE, the Ramtek code resulting from calls to any FLIK module goes into the display list definition, not to the Ramtek.

(2) DL_CLOSE -- terminates the definition phase and completes the compile phase. This module automatically provides a Ramtek return instruction as the last word in the display list definition. DL_CLOSE writes the resulting local, common, and external symbolic tables, plus the actual display list code, to the appropriately named display-list file with file-type .DL_OBJ. If you have enabled a listing with DL_LISTING, then you also get a file of the same name but with file-type .DL_LIS, which you can type or see with the text editors.

(3) DL_LINK -- does the link stage, which is transparent to the user. You can also run a program called OFLIKDL:LINKER.EXE from VAX/VMS DCL to accomplish the same thing interactively. The application or the operator simply passes or enters a list of the display list files to link together, plus the desired output file name. From all the specified .DL_OBJ files, DL_LINK merges their common and external symbol tables, merges their common-area requirements, and finally combines their display list code and total common areas into contiguous sections, each earmarked for a specific Ramtek memory segment. DL_LINK saves the merged symbol tables and all of these sections onto a single loadable file, with type .DL_EXE. If you requested a listing with DL_LISTING, FLIK also writes out a .DL_MAP file which you can type or see with the text editors.

(4) DL_SEND -- loads the single .DL_EXE file specified by the user into Ramtek memory, and keeps the symbol tables on the host side so that applications can call the loaded display lists with their symbolic name. DL_SEND can reference the Ramtek common area variables by their symbolic name.

(5) DL_CALL -- calls a display list to run, using its external symbolic name (see paragraph G-4 below).

G-4 <u>FLIK symbolic names and symbol tables.</u> Display list programs on the Ramtek can do most normal graphics display operations, but they also can branch, load-store between memory and registers, and call other display lists. There is a "common" area of display-list memory set aside on the Ramtek for interdisplay-list communications. Without FLIK_DL, for a user programmer to take advantage of these extra Ramtek facilities required that he use absolute Ramtek memory addresses. With FLIK, the user can reference memory locations with three types of symbolic names: local, common, and external. Load-stores use local or common addresses. Branches and loops use local addresses. Calls to display lists use external addresses. FLIK builds symbol tables for these addresses and operates with them transparent to the user during the compile and link phases. FLIK makes the host privy to the common and external addresses to enable a host program to call a display list by its external symbolic name and to reference locations in the Ramtek common memory by their common symbolic name. These symbolic names are relocatable, so when a user changes and recompiles display list code or links together different combinations of display lists that call each other, the same symbolic names remain, even though the corresponding absolute addresses will change.

G-5. Starting a display list. There are two causes for a display list to execute. The more frequent cause is that either the host or another display list calls it. The less frequent cause is that the display list is entered in a local cursor or keyboard function table. Entry in the cursor or keyboard function table tells the the Ramtek itself to execute that display list when someone operates the Ramtek tablet, cursor controller, or keyboard. When using FLIK, these same Ramtek rules still hold true. However, FLIK provides you with calls for several situations, based on the particular stage of FLIK display list operations and on the manner in which the display list is to be invoked. The specific FLIK modules for each of these situations appear below. See paragraphs G-10 and G-15 for further information. Remember that in any case, the display list you want to execute must have been already compiled by FLIK between a DL_OPEN and a DL_CLOSE, linked by DL_LINK, and loaded by DL_SEND. (See paragraph G-3). Additionally, remember that instead of requiring absolute addresses to identify a display list, FLIK provides external symbolic names for that purpose.

a. During the definition and compile stage (between DL_OPEN and DL_CLOSE,) insert code into this display list to invoke another display list (or sometimes even itself.)

(1) DL_CDL -- between DL_OPEN and DL_CLOSE, inserts code into a display list to call another display list. This call will actually occur when that first display list is later executed.

(2) DL_SLCF and DL_CLCF -- between DL_OPEN and DL_CLOSE, DL_SLCF inserts code into a display list to place an entry into a local cursor function table for the execution of that or another display list. The placement of this entry will actually occur when this display list is later executed. DL_CLCF inserts an instruction that will remove this cursor function table entry.

b. After FLIK has defined and compiled .DL_OBJ files between DL_OPEN and DL_CLOSE, linked those .DL_OBJ files with DL_LINK, and sent the resulting .DL_EXE file to the Ramtek with DL_SEND, you can start a display list with a call or through a local cursor or keyboard function table. Remember: with FLIK, you reference a display list with its external symbolic name to call it under any and all conditions, even from the local function table. This is true whether at define and compile time or after loading the display list to the Ramtek.

(1) DL_CALL -- from the host, execute a display list immediately.

(2) DL_SET_LCF and DL_CLCF -- DL_SET_LCF will make a local cursor function table entry from the host, so that when someone operates the puck or cursor controller the Ramtek will execute that previously loaded display list. DL_CLCF also works from the host to remove this entry in the local cursor function table, put there either by DL_SET_LCF, or by a display list whose definition included Ramtek instructions produced by DL_SLCF.

G-6. <u>Returning from display lists.</u> For returns from a display list, FLIK automatically puts into the code one and only one return, at the bottom of the definition. If you want extra returns, for example just prior to an extra entry point, you call KRETDL.

G-7. <u>Entry points into display lists.</u> The define and compile stage automatically sets up an external symbol for the top of the display list, where the symbol name equals the display list name. You can create additional external symbols within the display list for extra entry points using the module DL_ENTRY, shown in the example below. This might produce a sequence as follows in the define and compile stage.

```
      CALL DL_OPEN ('ODLDATA:DRAW4')
CC    .      .     .
CC    .      .     .      MAIN PORTION OF 'DRAW4'
CC    .      .     .
CC    .      .     .
           CALL KRETDL
 5000      CALL DL_ENTRY ('DRAW4_END', )
CC    .     .
CC    .     .SECOND PORTION OF 'DRAW4' CALLED  'DRAW4_END'
CC    .     .
CC    .     .
      CALL DL_CLOSE
```

G-8. <u>Flow control.</u> In Ramtek display list code there is one command for flow control, the "jump on display list register" (JDLR) instruction. With this one instruction you can do conditional or unconditional branching to a specified absolute address. In any case you select one of 16 display list registers to test, and a type of test (such as unconditional branch, greater than zero, etc.). If the test succeeds, you branch.

a. FLIK display list capabilities allow you to use this same instruction in a similar way, but you supply parameters for the instruction that conform to the FLIK display list design philosophy. The address is a local symbolic name plus an optional word offset, not an absolute address. The type of test is defined by character-string mnemonics (like GOTO, LT, EQ, NE, GT, SET, RESET, etc.), not with an integer condition value. For people familiar with the JDLR this may take some getting used to.

b. FLIK also provides modules to create rudimentary IF-THEN/ELSE/ENDIF and DO/ENDDO constructs in display lists. You can have nested IF-THEN/ELSE/ENDIF constructs and nested DO/ENDO constructs, and nest IFs within DOs, and vice versa. There is no DO WHILE, and the IF test can only have one predicate.

(1) IF-THEN/ELSE/ENDIF construct. The IF predicate specifies an execution time comparison between two operand values. The first of those two values is always specified in the IF predicate as a display-list register number. At execution time the contents of that display-list register will be the first operand in the comparison. There, the IF predicate provides you with two possible ways to specify the second operand in the comparison. One way to specify the second operand is to provide another display-list register number. At execution time the contents of that other display-list register will be the second operand in the comparison. The other way to specify the second operand is to provide a constant value. At execution time that constant will be the second operand in the comparison. You use mnemonics such as LT, LE, EQ, etc., to specify the type of comparison. The benefit of this method over the JDLR is that FLIK does some of the necessary manipulations for you that you would ordinarily do to achieve a certain test. Also, you need not deal with any absolute addresses or even the FLIK local branch symbolic labels to build IF-THEN/ELSE/ENDIF constructs. FLIK automatically provides these labels for you.

(2) DO-ENDDO construct. The DO/ENDDO provides the usual do loop construct: I = E1, E2, E3, where I is an increment value, E1 is the starting value, E2 is the ending value, and E3 is an increment. The increment can be either positive or negative. The loop test is at the top of the loop. The three loop controls can either be determined at execution time from the contents of display-list registers, or they can be constants. This is similar to the predicate operands in the IF-THEN/ELSE/ENDIF construct. The benefit of using the FLIK DO/ENDDO is that you do not have to write that display-list code yourself every time you need it, and you need not work with any local symbolic labels or absolute addresses for the top and bottom of each loop. FLIK automatically supplies local symbolic labels for those points in the code. When using the DO constructs you need to remember that you have "reserved" registers for the increments in the DO. When nesting DO loops be sure that each loop uses a different register for the increment value. FLIK does not check for this.

G-9. Interactive Ramtek input. The Ramtek display list facility contains an instruction to take input from Ramtek peripherals such as keyboard, cursor controller, tablet, etc. FLIK will insert this instruction into a display list during the define and compile phase. To do so, you call DL_PLDLR. You specify a peripheral device, the type of input from that device, and the Ramtek display list registers into which to transfer the data. After that call, you can define other instructions in the display list code that will test those input data or use coordinates or ASCII characters for graphics drawing, etc.

G-10. Host-Ramtek synchronization. The host normally controls the execution of Ramtek code through display list calls. The Ramtek can bring the host out of a wait state by generating interrupts resulting from a Ramtek user operating the tablet or keyboard. This always happens without a cursor function table entry. But, when working with local function table entries, you can use a flag (H-BIT) contained therein to enable or disable that host interrupt. From within a display list, the Ramtek itself can change this table entry including the H-BIT. So with a local function table in use, and the host waiting for an interrupt from the Ramtek, the Ramtek can use the H-BIT for partial control over the time that the host interrupt will occur. However, the Ramtek still depends on activity at its peripherals. That activity combined with the H-BIT enabled will interrupt the waiting host.

a. We can illustrate this with lines from the etch-sketch example below. We show lines from the define and compile of a two-section display list. It has two entry points. The main entry point is 'DRAW4', and the second entry point is 'DRAW4_END'. The code of 'DRAW4' has instructions to get the tablet and cursor coordinates and status. It also contains select cases to test for puck button that tell what action the user is taking. The button values are in a display list register whose number is set in IDLR_FLAG. We have omitted all select cases but one.

b. The select case shown is for the "quit" button test. If this test succeeds, then the code will change the cursor table--it will switch on the H-BIT and use a different entry point, 'DRAW4_END'. For the next tablet activity, the Ramtek will interrupt the host and execute at 'DRAW4_END'.

```
CC  THE VARIABLES IDLR_* ARE INTEGERS FROM 0 TO 15 AND SPECIFY
CC  RAMTEK DISPLAY LIST REGISTERS. USING THESE VARIABLES
CC  INSTEAD OR CONSTANTS MAKES IT EASIER IN A LARGE PROGRAM
CC  TO CHANGE THE USAGE OF THE SIXTEEN AVAILABLE RAMTEK
CC  DISPLAY LIST REGISTERS.
CC
        IDLR_FLAG = 0  ! USE FOR TESTING TABLET BUTTONS
        IDLE_CURS = 5  ! THIS PLUS NEXT TWO REG FOR X,Y,STAT
        IDLR_TAB  = 8  ! THIS PLUS NEXT TWO REG FOR X,Y,STAT
```

```
      ITABNUM   = 0  ! HARDWIRE TO CURSOR ZERO.
CC
      CALL DL_OPEN ('ODLDATA:DRAW4')
CC                LOAD INTO THREE REGISTERS THE CURRENT
CC                CURSOR X, Y, AND STATUS.
          CALL DL_PLDLR (IDLR_CURS, 'CURS', ITABNUM)
CC                LOAD INTO THREE REGISTERS THE CURRENT
CC                TABLET X, Y, AND STATUS.
          CALL DL_PLDLR (IDLR_TAB, 'TAB', ITABNUM)
CC    .     .
CC    .     .CODE TO MOVE THE LOW FOUR BITS OF TABLET
CC    .     .STATUS INTO 'IDLR_FLAG' REGISTER.
CC    .     .SELECT-CASES IN 'DRAW4' TO HANDLE PICTURE
CC    .     .DRAWING.
CC    .     .
CC  SELECT CASE -- QUIT (LOW-FOUR-BITS OF TABLET STATUS = 4)
          CALL DL_IF   (1, IDLR_FLAG, 'EQ', 4)
CC             CHANGE THE LOCAL CURSOR FUNCTION TABLE ENTRY
CC             TO INCLUDE THE H-BIT, AND TO EXECUTE STARTING
CC             AT 'DRAW4_END'.
               CALL DL_CLCF (ITABNUM)
               CALL DL_SLCF ( 1, ITABNUM, ICONTEXT, , 'DRAW4_END')
          CALL DL_ENDIF
CC             RETURN FROM ALL SELECT-CASES.
          CALL KRETDL
 5000     CALL DL_ENTRY ('DRAW4_END', )
CC                 INTERRUPT COMES HERE WHEN H-BIT IS SET.
CC                 TURN OFF THE CURSOR USING INVISIBILITY AND
CC                 PLACEMENT OFF-SCREEN.
          CALL KWCURS ( 0, 0, 640, 512)
      CALL DL_CLOSE
```

(1) We pulled these lines from the software that uses the etch-sketch display list. After sending it, this code enables local cursor function for the main entry point 'DRAW4', but without the H-BIT. Then, the code waits for an interrupt from the Ramtek, which under present conditions will never happen.

(2) At this point, the user can sketch a picture without interrupting the host. When the user finishes his picture, he presses the blue puck button meaning "quit". In the code lines shown above for the 'DRAW4' definition, we show the "quit" select case. As described there, etch-sketch itself alters the local cursor function table to include host interrupt and the execution of not 'DRAW4' but 'DRAW4_END'. The next tablet activity executes 'DRAW4_END' and interrupts the host. The display list turns off the cursor, and this host program disables local cursor functions. The lines omitted here are the reading back of the picture to the host. They appear below in the actual program example. This is just an outline.

```
CC              DISABLE LOCAL CURSOR FUNCTIONS AND CLEAR THE
CC              TABLE.
        CALL KSETLCFS (ITABNUM, 1)
        CALL DL_CLCF (ITABNUM)
CC              SEND THE ETCH-SKETCH DISPLAY LIST TO RAMTEK.
        CALL DL_SEND  ('ODLDATA:DRAW4', NDL, DL_LIST)
CC               SET THE TABLET TO INTERRUPT IN TWO CASES:
CC               TK = WHEN USER HOLDS DOWN AT LEAST ONE BUTTON AND
CC                    MOVES THE PUCK.
CC               TE = WHEN USER RELEASES THE LAST OF ONE OR MORE
CC                    BUTTONS THAT WERE DEPRESSED.
        ISTAT = '5000'X + '0100'X + + '0F'X
                                    ! (TE+TK) + TD + (F3+F2+F1+F0)
        CALL KWTABST ( ITABNUM, ISTAT, 4, 4)
CC              MAKE THE CURSOR VISIBLE.
        ISTAT = 1
        CALL KWCURS ( ITABNUM, ISTAT, 640, 512)
CC          ENABLE CURSOR INTERRUPTS ON RAMTEK FOR THAT TABLET.
        CALL KSETLCFS (ITABNUM, 2)
CC          SET THE LOCAL CURSOR FUNCTION FOR THIS TABLET
CC          SO THE RAMTEK, WHEN SOMEONE USES THE PUCK,
CC          WILL JUMP TO DL-ENTRY POINT 'DRAW4', AND EXECUTE
CC          THIS DISPLAY LIST.  TELL THE RAMTEK NOT TO
CC          INTERRUPT THE HOST WHEN THIS HAPPENS.
        IHBIT = 0
        CALL DL_SET_LCF ( IHBIT, ITABNUM, ICONTEXT,  , 'DRAW4')
CC          WAIT UNTIL RAMTEK SENDS BACK AN INTERRUPT.
        CALL KRTABST ( ITABNUM, IX, IY, ISTAT )
CC.           .
CC.           . THE CODE OMMITTED READS BACK THE PICTURE SKETCHED
CC.           .
CC.           .
CC.           .
CC            SWITCH OFF THE LOCAL CURSOR FUNCTION AND
CC            REMOVE ENTRY FROM TABLE.
        CALL KSETLCFS (ITABNUM, 1)
        CALL DL_CLCF (ITABNUM)
```

G-11. Hardware dependency of display lists. There are some Ramtek instructions, especially read-back instructions, that do not work in display lists, even in a FLIK environment. Refer to the Ramtek Software Reference Manual when in doubt about any given instruction. Also, FLIK produces Ramtek dependent display list code, tuned by FLIK to the attached Ramtek when compiling the display list. This ties display lists to Ramtek hardware configurations. With some planning you can minimize the resulting problems. FLIK normally adjusts to several Ramtek peculiarities: the resolution, the number of planes available, the allocation of those planes to the color/levels layout, the tablet and cursor numbers for the current station, etc. With FLIK, Ramtek instructions get rerouted into display list definitions. They carry with them the adjustments made by FLIK to the attached Ramtek station, while generating the display lists. In other words, the display list code becomes "tuned" to that Ramtek station hardware. To handle this problem, you have three courses of action: live with hardware-dependency of display lists and keep a set for each hardware station, remove the dependencies, or plan for the host program that calls the display lists to solve these dependencies either with Ramtek capabilities (such as the transformation matrix to solve the resolution problem), or with self-modifying display-list code. Self-modifying code is possible; for an example see paragraph G-14.

G-12. Basic example. The following example shows the use of all four operations of the FLIK display list package. The example includes a program that will define and compile three display lists that call each other. We also show the execution of a stand-alone display list linker. Then we show a second program that sends the linker output file to the Ramtek and calls the first display list which calls the second which calls the third.

a. Define and compile. The following application defines and compiles the three display lists.

```
      PROGRAM ELS
CC             DEFINE THE ARRAYS FOR FLIK COLOR/LEVEL
      PARAMETER NLVL = 1
      PARAMETER IDIM = 8
      INTEGER * 2 NPPL (NLVL)
      INTEGER * 2 ICOL_IN (IDIM, NLVL)
CC            THESE ARRAYS FOR VIRTUAL LINE DRAWING.
      REAL * 4 X (5), Y (5)
CC            SET UP THE FLIK COLOR/LEVEL DATA
      DATA NPPL /3/
      DATA ICOL_IN /'000'X,  ! BLACK
     *              'F00'X,  ! RED
     *              '0F0'X,  ! GREEN
     *              '00F'X,  ! BLUE
     *              'FF0'X,  ! YELLOW
     *              '0FF'X,  ! CYAN
     *              'F0F'X,  ! PURPLE
     *              'FFF'X / ! WHITE
CC            SET UP THE RECTANGLE VIRTUAL CORNERS.
      DATA X /1000., 10000., 10000.,  1000.,  1000./
      DATA Y /1000.,  1000., 10000., 10000.,  1000./
CC            INITIALIZE WITH A CERTAIN RAMTEK
      CALL KRMINIT (1280, 1024, 4, MON)
CC            SEND THE COLOR/LEVEL SETUP TO FLIK.
      CALL KOVRDEF (NLVL, NPPL, ICOL_IN, IDIM)
CC             DUMP ANY REMAINING CONTENTS IN THE FLIK
CC             OUTPUT BUFFER.
      CALL KFLUSH
CC             SWITCH ON THE DISPLAY-LIST LISTING OUTPUT.
      CALL DL_LISTING (.TRUE., .TRUE.)
CC             BEGIN DEFINITION AND COMPILE OF DISPLAY
CC              LIST NAMED 'ELCALL1' TO GO IN A FILE
CC              UNDER LOGICAL ODLDATA.
      CALL DL_OPEN ('ODLDATA:ELCALL1')
CC                  UNTIL FURTHER NOTICE,
CC                  SUBSEQUENT FLIK CALLS PRODUCE RAMTEK CODE
CC                  THAT GOES INTO DISPLAY LIST ELCALL1.
          CALL KOVRBUF (1)
          CALL KOBCOLR (1)
          CALL KOCOLOR (2)
CC                  ERASE PLANES WITH SPECIFIED COLOR.
          CALL KOERASE (1)
          IXCEN = 500
          IYCEN = 500
CC                 DRAW AND FILL A CIRCLE.
          CALL KCIRCLE (IXCEN, IYCEN, 200)
          CALL KOBCOLR ( 2)
          CALL KOCOLOR (2)
          MODE = 2
          CALL KFILL (MODE, IXCEN, IYCEN)
CC         SET VIRTUAL-TO-RAMTEK WINDOW MAPPING.
          HIGH = 10240.
          WIDE = 12800.
          CALL KCORDTRAN (0., 0., WIDE, HIGH, 0, 0, 1280, 1024)
CC          DRAW A BOX BORDER IN VIRTUAL
          CALL KOCOLOR (3)
          CALL KLADRAWS (X, Y, 4)
          WRITE (6, *) 'EL: after KLADRAWS'
CC              DRAW TEXT-NAME OF THIS DISPLAY LIST TO
CC              INDICATE FOR TEST THAT IT WAS CALLED.
          CALL KOCOLOR (4)
          CALL KTEXT (20, 20, 6, %REF ('EL-1'), 4)
CC              CALL DISPLAY-LIST ELCALL2.
CC              THIS DISPLAY LIST DOES NOT EXIST YET,
CC              SO IT BECOMES AN UNRESOLVED EXTERNAL
CC              REFERENCE.
          CALL DL_CDL ('ELCALL2')
CC          END THE DEFINITION OF DISPLAY-LIST ELCALL1,
CC          COMPILE IT, AND SAVE ON DISK FILE ELCALL1.DL_OBJ.
CC          ALSO PRODUCE A LISTING ON ELCALL1.DL_LIS.
      CALL DL_CLOSE
      CALL KNBATCH
CC              BEGIN DEFINITION AND COMPILE OF DISPLAY
CC              LIST NAMED 'ELCALL2' TO GO IN A FILE
CC              UNDER LOGICAL ODLDATA.
      CALL DL_OPEN ('ODLDATA:ELCALL2')
CC                  UNTIL FURTHER NOTICE,
CC                  SUBSEQUENT FLIK CALLS PRODUCE RAMTEK CODE
CC                  THAT GOES INTO DISPLAY LIST ELCALL2.
          CALL KOVRBUF (1)
          CALL KOCOLOR (2)
          CALL KOBCOLR (1)
CC                 ERASE PLANES WITH A SPECIFIED COLOR.
          CALL KOERASE (1)
          IXCEN = 500
```

```
              IYCEN = 500
              IXAXIS = 100
              IYAXIS = 200
CC                   DRAW AND FILL AN ELLIPSE.
              CALL KELLIPSE (IXCEN, IYCEN, IXAXIS, IYAXIS)
              CALL KOBCOLR ( 2)
              CALL KOCOLOR (2)
              MODE = 2
              CALL KFILL (MODE, IXCEN, IYCEN)
CC             SET VIRTUAL-TO-RAMTEK WINDOW MAPPING DIDFFERENTLY.
              HIGH = 10240.
              WIDE = 12800.
              CALL KCORDTRAN (0., 0., WIDE, HIGH, 0, 0, 1280, 1024)
CC              DRAW A BOX BORDER IN VIRTUAL
              CALL KOCOLOR (3)
              CALL KLADRAWS (X, Y, 4)
              WRITE (6, *) 'EL: after KLADRAWS'
CC                  DRAW TEXT-NAME OF THIS DISPLAY LIST TO
CC                  INDICATE FOR TEST THAT IT WAS CALLED.
              CALL KOCOLOR (4)
              CALL KTEXT (120, 20, 6, %REF ('EL-2'), 4)
CC                  CALL DISPLAY-LIST EL3.
CC                  THIS DISPLAY LIST DOES NOT EXIST YET,
CC                  SO IT BECOMES AN UNRESOLVED EXTERNAL
CC                  REFERENCE.
              CALL DL_CDL ('EL3')
CC              END THE DEFINITION OF DISPLAY-LIST ELCALL2,
CC              COMPILE IT, AND SAVE ON DISK FILE ELCALL2.DL_OBJ.
CC              ALSO PRODUCE A LISTING ON ELCALL2.DL_LIS.
          CALL DL_CLOSE
          CALL KNBATCH
CC                 BEGIN DEFINITION AND COMPILE OF DISPLAY
CC                 LIST NAMED 'EL3' TO GO IN A FILE
CC                 UNDER LOGICAL ODLDATA.
          CALL DL_OPEN ('ODLDATA:EL3')
CC                     UNTIL FURTHER NOTICE,
CC                     SUBSEQUENT FLIK CALLS PRODUCE RAMTEK CODE
CC                     THAT GOES INTO DISPLAY LIST EL3.
              CALL KOVRBUF (1)
              CALL KOCOLOR (3)
              CALL KOBCOLR (1)
CC                   ERASE PLANES WITH A SPECIFIED COLOR.
              CALL KOERASE (1)
CC
              IXCEN = 100
              IYCEN = 500
CC                   DRAW AND FILL TWO CIRCLES.
              CALL KCIRCLE (IXCEN, IYCEN, 50)
              CALL KOBCOLR (3)
              CALL KOCOLOR (3)
              MODE = 2
              CALL KFILL (MODE, IXCEN, IYCEN)
CC
              IXCEN = 400
          IYCEN = 500
          CALL KCIRCLE (IXCEN, IYCEN, 50)
          CALL KOBCOLR (3)
          CALL KOCOLOR (3)
          MODE = 2
          CALL KFILL (MODE, IXCEN, IYCEN)
CC         SET VIRTUAL-TO-RAMTEK WINDOW MAPPING STILL
CC         DIFFERENTLY.
          HIGH = 10240.
          WIDE = 12800.
          CALL KCORDTRAN (0., 0., WIDE, HIGH, 0, 0, 1280, 1024)
CC          DRAW A BOX BORDER IN VIRTUAL
          CALL KOCOLOR (4)
          CALL KLADRAWS (X, Y, 4)
          WRITE (6, *) 'EL: after KLADRAWS'
CC              DRAW TEXT-NAME OF THIS DISPLAY LIST TO
CC              INDICATE FOR TEST THAT IT WAS CALLED.
          CALL KOCOLOR (4)
          CALL KTEXT (320, 20, 6, %REF ('EL-3'), 4)
CC          END THE DEFINITION OF DISPLAY-LIST EL3,
CC          COMPILE IT, AND SAVE ON DISK FILE EL3.DL_OBJ.
CC          ALSO PRODUCE A LISTING ON EL3.DL_LIS.
      CALL DL_CLOSE
      CALL KNBATCH
CC
      STOP
      END
```

You enter, FORTRAN-compile, link, and run this program. The link must include OFLIKDL:A/LIB, and OKGL:FLIK/OPT. When you run the program and initialize with a certain Ramtek station, FLIK will define and compile the three display lists, tailored to fit on the selected station. The run produces three display list files on ODLDATA, namely ELCALL1.DL_OBJ, ELCALL2.DL_OBJ, AND EL3.DL_OBJ. There are also a similar set of .DL_LIS files that you can type or edit.

b. Off-line linker. Next, to link the three display lists you run the off-line display-list linker program as follows.

```
$ RUN OFLIKDL:LINKER
MAKE list of DL's to link
DL-name, or <CR> to end ODLDATA:ELCALL1
DL-name, or <CR> to end ODLDATA:ELCALL2
DL-name, or <CR> to end ODLDATA:EL3
DL-name, or <CR> to end <cr>
ENTER DL-EXE file name ODLDATA:ELS
output a listing? [Y/N] [N] Y
DL_DLLINK: link phase
DL_DLLINK: commons  phase
DL_DLLINK: positioning phase
DL_DLLINK: segment phase
DL_LINK: save-EXE  phase
```

```
DL_DLPREP: external  0047 4100 8000
DL_DLPREP: external  0039 4100 8094
LOADED DLS
FORTRAN STOP
```

c. Send and call. Finally, with another program you load and call the display lists in the following piece of code. For simplicity, this sample program is noninteractive: the name of the DL_EXE file to load and the name of the display list on that file to call are both hardwired. In the real version of this software, available as OFLIKDL:RUNNER.EXE, you respond to prompts. First, you enter the name of the .DL_EXE file and see a list of the display lists it contains. Then, you enter the name from that list of the display list now loaded which you wish to execute.

```
      PROGRAM RUNNER
CC
      CHARACTER  * 64 DL_LIST (64)
      CHARACTER * 1 YESNO
      CHARACTER  * 12 DL_SEL
      CHARACTER * 80 DL_EXE_NAME
CC
      LOGICAL SEPARATE_LOAD
      LOGICAL ALREADY_IN_RAMTEK
CC              SAME COLOR/LEVEL DEFINITIONS AS IN THE
CC              PROGRAM THAT DEFINED AND COMPILED THE
CC              DISPLAY LISTS. THIS IS CRITICAL IF YOU
CC              INSIST ON SETTING COLORS AND LEVELS IN THE DL.
      PARAMETER NLVL = 1
      PARAMETER IDIM = 8
      INTEGER * 2 NPPL (NLVL)
      INTEGER * 2 ICOL_IN (IDIM, NLVL)
CC
      REAL * 4 X (5), Y (5)
CC
      DATA NPPL /3/
      DATA ICOL_IN /'000'X,  ! BLACK
     *              'F00'X,  ! RED
     *              '0F0'X,  ! GREEN
     *              '00F'X,  ! BLUE
     *              'FF0'X,  ! YELLOW
     *              '0FF'X,  ! CYAN
     *              'F0F'X,  ! PURPLE
     *              'FFF'X / ! WHITE
CC             INITIALIZE THE RAMTEK.
      CALL KRMINIT (1280, 1024, 4, MON)
CC             SET THE SAME COLORS/LEVELS AS BEFORE.
      CALL KOVRDEF (NLVL, NPPL, ICOL_IN, IDIM)
CC              TURN ON THE LISTING OPTION TO PRODUCE
CC              A .DL_MAP FILE.
      CALL DL_LISTING (.TRUE., .TRUE.)
CC              SEND THE DISPLAY LIST LINKED FILE THAT
CC             YOU JUST LINKED OFFLINE.
      CALL DL_SEND ('ODLDATA:ELS', 1, DL_LIST)
CC              SHOW USER WHICH DISPLAY LISTS WERE SENT.
      WRITE (6, *) 'SENT ', NDL, ' DLs'
      DO IDL = 1, NDL
          LAST = LASTNONB (DL_LIST (IDL))
          WRITE (6, '(1X, A)') DL_LIST (IDL) (1 : LAST)
      ENDDO
CC                 CALL THE TOP LEVEL DISPLAY LIST THAT
CC                 WILL CALL THE SECOND THAT WILL CALL
CC                 THE THIRD.
      CALL DL_CALL (DL_LIST ('ELCALL1')
      STOP
      END
```

G-13. Register instructions. All these instructions involve the 16 display list registers provided by the Ramtek. Some of these instructions also involve locations in display list memory.

a. Ramtek register operations. The original Ramtek (not FLIK) instructions for several types of register operations appear below.

(1) Set display list register (SDLR): Set a display list register to a constant value.

(2) Load (LDLR) and store (STDLR): Move data between a display list register and memory, or vice versa. You specify the memory address--absolute or immediate. With absolute addressing, the actual address of the memory location is in the instruction. With immediate addressing, the absolute address of the memory location is expected to be in another specified display list register at execution time.

(3) Binary operators. Perform arithmetic and Boolean operations between two operands. One is always a display list register, and the other can either be a display list register or a constant.

b. FLIK register operations. FLIK provides capabilities similar to those shown above. In addition, FLIK provides not one but three load-store modules, depending on the type of addressing you need.

(1) Set display list register (SDLR). Both Kellner and FLIK provide the regular SDLR instruction with module KSSETDLR. FLIK also allows you to set the contents of a display list register equal to the absolute address corresponding to a local symbolic name.

(2) Memory location label (DL_LABEL). You can assign a local symbolic name to the current position (with optional offset) in the display list code definition. Use these labels

for all local symbolic memory accesses used by load-store and branch operations.

(3) Load-store (DL_MEM). You can move a word between a display list register and a local or common symbolic location (with optional constant offset).

(4) Load-store relative (DL_MEMR). You can move a word between a display list register and a local or common symbolic location with a displacement being the execution time contents of a display list register.

(5) Load-store immediate (DL_MEMI). You can move a word between a display list register and a memory location whose absolute address resides at execution time in another display list register

(6) Register-to-register operations (DL_R2R). You can apply arithmetic or Boolean operations to two operands; the first is always a display list register, and the second is either a display list register or a constant value.

G-14. Ramtek common. FLIK provides capabilities to access the Ramtek "common" area. For background information on Ramtek memory segmentation and allocation, absolute addressing, the Ramtek common, and the FLIK linker, see below.

a. Segments, allocation, and absolute addresses. The Ramtek itself provides 16 segments of display list memory, where one segment is the special "common" area and the other 15 are the normal segments provided to contain display list programs. Note that the extended memory option increases the available display list memory. Before it is used, a normal segment must be allocated to contain from one to four blocks of physical memory, 4096 bytes each. The block size of the accessable "common" area varies as explained below. A single normal segment may contain one or more display lists. All display lists in a given segment access display list memory with a range of address values from 8000 to BFFF hexadecimal (hex), thereby including all four possible 4096 byte blocks. Addresses from 8000 to 8FFF hex access memory in the first block, from 9000 to 9FFF hexadecimal access the second block, from A000 to AFFF hex access the third block, and B000 to BFFF access memory in the fourth block. The Ramtek provides that same address range to display lists working in any of the 15 mal segments.

b. Common area size and access. The "common" area always has four 4096 byte blocks. Omission of any of the four possible blocks from a normal segment's allocation allows display lists in that segment to access those corresponding blocks in the "common" area. Therefore, display lists in a three-block normal segment with addresses 8000 to AFFF hex can access the highest block of "common" with addresses B000 to BFFF hex; display lists in a two block normal segment with addresses 8000 to 9FFF hex can access the two highest blocks of "common" with addresses A000 to BFFF hex; and display lists in a one-block normal segment with addresses 8000 to 8FFF hex can access the three highest blocks of "common" with addresses 9000 to BFFF hex. The most "common" area available to display lists loaded in a normal segment is three blocks, from 9000 to BFFF hex.

c. FLIK handling of segment and Ramtek "common". The FLIK logic that links display lists and handles Ramtek "common" conforms to the addressing rules stated above. Provided with an input list of display lists, the FLIK linker makes several passes through that list. One pass builds a "common" area usage table containing the size of all required "common" area spaces, the variables those spaces contain, and which display list(s) require them. These spaces are analagous to FORTRAN named common areas and are a FLIK, not a Ramtek, concept. A subsequent pass of the linker builds segments for subsequent loading in the Ramtek display list memory. It builds one or more normal segments from the display lists named in the input list, and builds the "common" area segment from the table of required "common" area spaces. The linker puts each display list that accesses Ramtek "common" into a normal segment short enough to allow that display list to access addresses within the block(s) of "common" area that contain the "common" area spaces required by that display list.

d. FLIK user-level "common" area facilities. With FLIK, a display list accesses the Ramtek "common" area using symbolic names for "common" area spaces and the variables within them. Several display lists included in the same FLIK display-list link can intercommunicate using the same "common" space and variable names resulting in access to the same absolute addresses within the Ramtek "common" area. In addition, those same display lists and the host program that "sent" them can communicate with each other through Ramtek common areas using those same symbolic "common" space and variable names. To prepare for and use these facilities, the application must do the following things in the order shown and in the proper stage of display list operations.

(1) During the define and compile phase, call DL_COMMON. When called during the definition phase, this module defines a common area and the spaces within it to be accessed by the current display list. You supply a symbolic common name, a symbolic variable name, a length, and optional initial value.

(2) After the send phase, call DL_COMMON again. Do so EXACTLY as you did during the define and compile phase with the EXACT same arguments. This time, you define a common area and the spaces within it to be accessed by the host program.

(3) After the send phase and after the DL_COMMON calls following the send phase, then call DL_LAYOUTCMN. DL_LAYOUTCMN completes the common symbol table information for use on the host side.

(4) Either before or after the call to the display lists that use the common areas defined, you can call DL_ACCESSCMN. This module lets you transfer information in either direction between a host application buffer and Ramtek common. The user specifies the direction of transfer, the common area symbolic name, the symbolic-name variable within that area and a word offset, a number of words to transfer, and a host buffer. Note that you can transfer several contiguous Ramtek common variables with a single call to DL_ACCESSCMN by using a word count set to the total of the size of all those variables.

> WARNING: The order of variables within Ramtek common memory is the reverse of the order you declare them with the DL_COMMON call. Watch out for this when transferring several variables at once between host and Ramtek. The easiest way to handle this is to reverse the order of the DL_COMMON calls when defining and compiling and later after the "send" phase.

These facilities allow the host to supply display lists with data before calling them, and to read back data produced by a display list after the display list has executed.

G-15. FLIK display list modules. A brief summary of the FLIK display list modules appears below.

a. Control. These modules determine the display list system stage of operation. Those stages are: define and compile, link, send (load), and call.

(1) DL_LISTING (ONOFF, FILE): Enables/disables debug listing to a file during compile, link, or load stages.

(2) DL_OPEN (DL_NAME): Starts the definition and compile of a display list.

(3) DL_CLOSE: Ends the definition of a display list, finish compiling it, and save it on a .DL_OBJ file.

(4) DL_LINK (LEN_LIST, DL_LIST, DL_EXE_NAME): Links together one or more display list .DL_OBJ files to one .DL_EXE loadable file.

(5) DL_SEND (DL_EXE_NAME, LEN_LIST, DL_LST): Loads the display lists contained on one .DL_EXE file to Ramtek memory segments.

(6) DL_LOAD (LEN_LIST, DL_LST, ALREADY_IN_RAMTEK): Links one or more display list .DL_OBJ files and loads them into Ramtek. However, do not produce a DL_EXE file.

b. Flow control. These modules work during the define and compile stage, between DL_OPEN and DL_CLOSE, to insert instructions into the display list being defined. These instructions are flow-control instructions for calling other display lists and for branching within the display list being defined.

(1) DL_CDL (SYM): Inserts a CDL instruction to call another display list into the display list definition.

(2) DL_ENTRY (SYM, IOFF): Inserts no instruction, but marks a place in this display list definition as an external symbol, to use as an entry point from outside this display list.

(3) DL_DO (IBIT1, IBIT2, IBIT3, INDX, I1, I2, I3): Inserts a DO-ENDDO construct into the display list definition. Nesting of DO-ENDOs is permitted. The first three arguments describe the last three arguments. This construct requires you to specify a display list register to use as the loop index. The loop has the usual three control parameters--initial, end, and increment, specified as last three arguments. Any of those three parameters can be either constants or contained at run time as indicated by the first three arguments. Note that the increment parameter value can be negative as well as positive.

(4) DL_ENDDO: Places the bottom of the innermost DO-ENDDO construct currently open.

(5) DL_IF (IBIT, IDLR, CNDX, IP2): Inserts the start of an IFTHEN-ENDIF or IFTHEN-ELSE-ENDIF construct into the display list definition. Similar to the DO-ENDDO, the first argument describes the last argument. This construct tests a display list register's contents at run time against the last argument, using the specified condition. The last argument can be either a constant or the contents at run time of another display list register, as indicated by the first argument. is allowed.

(6) DL_ELSE: Inserts the ELSE portion of an IFTHEN-ELSE-ENDIF construct into the display list definition.

(7) DL_ENDIF: Inserts the ENDIF portion of either the IFTHEN-ELSE-ENDIF or IFTHEN-ENDIF construct into the display list definition.

(8) DL_JDLR (IDLR, IBIT, COND, SYM, IOFF): Inserts the standard Ramtek jump on display list register instruction into the display list definition. This includes the specified register, the bit selector option, the type of test, the local label to branch to, and an optional offset from that label.

(9) DL_LABEL (SYM, IOFF): Marks a place in the display list definition with a local label.

c. Code. These modules also insert Ramtek code into display list definitions but they do not pertain to branching or calling other display lists as did the modules in the previous section.

(1) DL_MEM (FCT, IDLR, SYM, IOFF): Inserts a load display list register, or a store display list register instruction into the display list definition. The first argument specifies "LDLR" or "STDLR". You use a local or common-area symbol and an optional offset to specify the memory location involved.

(2) DL_MEMR (FCT, IDLR, SYM, IDLR_OFF, IDLR_ADDR): Similar to the above, inserts a load or store instruction into the display list definition. The difference is that the offset from the local or common-area symbol is calculated at run time and saved in a second display list register. You need to supply a third display list register to use as a scratch for calculations.

(3) DL_MEMI (FCT, IDLR, IADRS_DLR): Similar to the above, inserts a load or store instruction into the display list definition. However, in this case, at run, time the memory address is contained time in the second register specified.

(4) DL_SDLRMEM (IDLR, SYM, IOFF): Inserts a set display list instruction into the display list definition. The value to set into the specified display list register is (after compiling and linking) the absolute address of the specified local or common-area symbol, optional offset included.

(5) DL_R2R (FCT, IBIT, IR, IR2, IR3): Inserts one of the many register-to-register instructions (such as "add display list register", "exclusive-or display list register", "multiply display list register", etc.) into the display list definition. The function argument determines which instruction to insert. All these instructions have two modes of operation, specified by IBIT. Ramtek calls this the "immediate" bit. When IBIT=0, this sets the register specified by IR to the result of applying the specified operation to the registers specified by IR2 and IR3. When IBIT=1, this sets the register specified by IR to the result of applying the specified operation to IR and the constant (NOT THE REGISTER) specified by IR2.

(6) DL_DATA (SYM, ISIZE, INIT): Reserves space at the end of the display list definition for local variables. You specify the symbolic name, the number of words, and a value to place in all those words.

(7) DL_SLCF (HBIT, IDEVICE, ICONTEXT, IFONT, SYM): Inserts a "set local cursor function table" instruction into the display list definition. This instruction, when later executed, will make an entry in the local cursor function table. That entry will tell the Ramtek to respond to cursor or puck activity by executing instructions in display list memory, starting at a

specified memory location. There is also an option to notify the host at that time. HBIT controls that option. IDEVICE specifies which cursor or tablet to use. ICONTEXT specifies the Ramtek context. IFONT specifies the font to use. SYM specifies the external FLIK display list entry point name for the starting point in display list memory.

(8) DL_COP (IX, IY): Prepares an (x,y) coordinate for inclusion in a set instruction to subsequently be inserted into the display list definition.

(9) DL_SIZ (XSIZE, YSIZE): Prepares a width and height size for inclusion in a set instruction to be inserted into the display list definition.

(10) DL_SET: Inserts the prepared parameters specified above into the display list definition.

d. Multistage modules. Each of the modules shown below works either during all display list stages or during two of those stages--the define and compile stage and the call stage. When used during the define and compile stage, each of these modules inserts instructions or otherwise contribute to the display list definition. The same modules used in other stages make no such contribution but have another effect.

(1) DL_COMMON (CMN_NAME, SYM, ISIZE, INIT): Using symbolic names, defines a variable in a named Ramtek "common" space. This module works during the definition stage and after the send stage. The variables fill the "common" space from high to low address. You specify the number of words in the variable and have the option to initialize all those words to a nonzero value.

(2) DL_DDLR (IBIT, IDLR): Either inserts into the display list definition, or executes immediately, an instruction to decrement the specified display list register by a value of one or two.

(3) DL_IDLR (IBIT, IDLR): Either inserts into the display list definition, or executes immediately, an instruction to increment the specified display list register by a value of one or two.

(4) DL_PLDLR (IDLR, FCT, IDEV): Either inserts into the display list definition, or executes immediately, a "parameter load display list register" instruction. This instruction places the specified information into one or more display list registers. Most of these data come from a cursor or tablet.

(5) DL_CLCF (IDEV): Either insert into the display list definition, or execute immediately, a "clear local cursor function table" instruction. This instruction removes the entry in the local symbol table for the specified device.

e. Post send stage. These facilities work only after sending a file of linked display lists to the Ramtek.

(1) DL_LAYOUTCMN: You call this module after one or more calls made, post send stage, to DL_COMMON. This module completes the tables started by DL_COMMON that describe the "common" area spaces on the Ramtek. Remember, you must be sure that you use the exact same DL_COMMON calls here as you did during the definition and compile stage.

(2) DL_CALL (SYM): Starts the execution of a display list contained in a file of linked display lists already sent to the Ramtek.

(3) DL_ENTRYADDR (SYM, IABS, ISEG, IERR): Host gets the absolute byte address and segment number of an external entry point within an already-sent display list.

(4) DL_ACCESSCMN (FCT, AREA, LCLSYM, IOFF, IBUF, LENW, IERR): Host transfers data in either direction between word(s) in a pre-loaded Ramtek named commons and a host buffer. You specify the common and starting word within it by a symbolic common and variable name and a word offset. You specify a host buffer variable and a number of words to transfer. If you are moving data with one of these calls for more than one symbolic variable in Ramtek common,

> WARNING: The order of variables in Ramtek common is backwards from the order that you declare them using the DL_COMMON calls. For example, two five-word variables named ALPHA and BETA would reside from low to high word in Ramtek common as BETA 1 through BETA 5, then ALPHA 1 through ALPHA 5. If you transfer 10 words starting at ALPHA 1 back to the host, these will not include any of BETA. You would need to transfer 10 words starting at BETA 1 to pick up all of BETA and ALPHA. You could instead make two calls to DL_ACCESSCMN, one for five words in ALPHA, and the other for five words in BETA.

(5) DL_ADDR (AREA, LCLSYM, IOFF, IADR, ISEG, IERR): Host gets the absolute byte address and segment number of a local variable in Ramtek common.

(6) DL_ALCON (ICONTEXT): Allocates a context.

(7) DL_DECON: Deallocates a context.

(8) DL_SET_LCF (HBIT, IDEVICE, ICONTEXT, IFONT, SYM): At run time executes a "set local function table" instruction to make an entry in the local cursor function table. When someone

uses the specified cursor or puck, this entry tells the Ramtek to execute display list instructions starting at the specified point. There is also an option that controls whether or not the Ramtek should notify the host when this occurs. HBIT controls that option. IDEVICE picks the cursor or tablet to respond to. ICONTEXT selects the context to switch to during execution of the display list, and IFONT selects the font type to use during that time. SYM is the external FLIK display list entry point label at which display execution is to begin when the user operates the puck or cursor.

f. Internal modules. There are additional modules contained within the FLIK-DL system that we do not show here. Those are lower level modules that operate during the various stages to help define, compile, link, or load display lists. They are not meant to be called by the application.

G-16. Etch-sketch example. The following example can be very complicated. We suggest that you familiarize yourself with paragraph G-15 of this document and with the Ramtek Software Reference Manual.

a. Major etch-sketch functions. The major function of etch-sketch allows you to use the puck to draw a picture containing lines, circles, and filled areas, to change drawing colors, and to end a picture. Etch-sketch saves every tablet-puck action as word-triples in Ramtek common. The host application that sent etch-sketch to the Ramtek waits in a tablet read for the picture entry to finish. When it does, the host application can read back from Ramtek common all the word-triples for your tablet-puck inputs that produced the picture. The host application can save those tablet-puck actions on disk; thus, another application can subsequently play back the picture you entered.

b. Special features. The etch-sketch display list contains several special features not found in ordinary display lists. These include display list activation through entries in the local cursor function table, modification of those entries by the display list itself to change which of its own sections is activated and to interrupt the host only when appropriate, interpretation of puck chords, and self-modification of display list code. We explain each of these special features in detail below.

(1) We designed this etch-sketch display list to execute differently than does a normal display list, using the concepts described in paragraph G-10. Rather than being called by another display list or the host, the etch-sketch code runs in response to tablet actions via the local cursor function table. Etch-sketch has two main sections each with its own entry points and corresponding external symbolic name. Only one of those entry points goes into the local cursor function table at any given time. When someone operates the tablet puck the Ramtek

starts execution of etch-sketch at its entry point currently set in the table. The normal etch-sketch table entry has the H-BIT reset so no host interrupts occur when etch-sketch executes. Etch-sketch has the ability to change which of its sections executes as a result of puck operation and to enable the host interrupt for that execution. To make this change, etch-sketch clears the current entry in the local cursor function table, then sets a new entry containing the address of the new entry point in etch-sketch and sets the H-BIT in that entry. Then, on the next puck action, the Ramtek will execute that different section of etch-sketch and interrupt the host.

(2) A tablet action starts the execution of etch-sketch through the local cursor function table and consists of (x,y) coordinates and a status including settings of the four puck buttons. Etch-sketch reads these data into display list registers using the parameter load display list register (PLDLR) instruction, then converts the four puck button settings (the low four status word bits) into a value from zero to 15. value selects the etch-sketch function for this puck action.

(3) Use of the puck buttons is somewhat unusual. You use them either individually or in "chords" to specify the type of etch sketch function--move, draw, dash, color up or down, fill, circle, etc. Since there are more than four etch-sketch functions we had to go to these "chords".

(4) Etch-sketch normally works with the tablet in "trailing-edge only" mode where a tablet action occurs only when you release the last puck button that had been depressed. Use of chords requires this approach. For some functions, however etch-sketch requires the tablet to operate in "track plus trailing-edge" mode. Along with the usual tablet actions from final puck button releases, this mode generates a stream of actions as the user holds down the puck button(s) and slides the puck across the tablet surface. During its execution, the etch-sketch display list can issue set tablet mode (STM) instructions to change between "trailing-edge only" mode and "track plus trailing-edge" mode. It does so when switching in and out of functions requiring the "track plus trailing-edge" mode.

(5) Etch-sketch performs the graphics commands you enter with tablet puck actions. To do so requires that etch-sketch execute standard Ramtek graphics instructions. However, with the tablet puck, you have specified some of the necessary parameters (such as coordinates, colors, etc.) that those standard Ramtek graphics instructions require to be set prior to their execution. To meet that requirement, etch-sketch must execute some self-modification code. The instructions requiring modification reside in known locations in display list memory. (FLIK allows local symbloic names for them). The self-modification consists of storing the required values into those locations. After those store operations, etch-sketch can execute the dynamically

modified standard Ramtek graphics instructions that now contain the values for coordinates, color, etc, that you entered with the tablet puck.

c. Etch-sketch common. This is a file of FORTRAN executable code to include in both of the following programs. It contains the Ramtek common-area description required by both the etch-sketch display list and the host program that reads back the sketched picture. The common includes room for current (x,y), foreground color, current etch-sketch action, maximum allowed size of the space for word-triples, and the large area to receive your tablet inputs, in the form of three words each.

```
CALL DL_COMMON ('C1', 'COPX',        1, 0)
CALL DL_COMMON ('C1', 'COPY',        1, 0)
CALL DL_COMMON ('C1', 'JSAVE',       1, 0)
CALL DL_COMMON ('C1', 'FGCOL',       1, 0)
CALL DL_COMMON ('C1', 'IFUNC',       1, 0)
CALL DL_COMMON ('C1', 'IPTR',        1, 0)
CALL DL_COMMON ('C1', 'MPTR',        1, 0)
CALL DL_COMMON ('C1', 'STAT_XY',  3000, 0)
                                ! ROOM FOR 1000 POINTS.
```

d. Logic flow. Etch-sketch contains two main sections--the main section and a termination section. The main section does a termination check, all tablet inputs, storage of input data, all changes of function, and all graphics display response. The termination section disables the tablet input and interrupts the host that should be waiting for a tablet input.

(1) The main etch-sketch code first checks for potential overflow of the Ramtek common where etch-sketch stores tablet-puck actions. If no space remains, etch-sketch branches to its "quit" code. Otherwise, the etch-sketch will process the current puck hit.

(2) As long as there remains room in the Ramtek common to save puck actions, the etch-sketch main code continues to execute. It gets the current tablet coordinates and status and determines the etch-sketch function selected by the puck buttons. It also gets the cursor coordinates. These tablet and cursor data not in display list registers and the current color and the saved actions in Ramtek common constitute the main data elements that etch-sketch manipulates.

(3) The main section of etch-sketch then compares the current action to the last one, to keep current the action label on screen, and to control the way the puck responds to the user. If this is a different action, etch-sketch draws the name of the new action in the upper left corner of the screen. If changing to the draw function, etch-sketch enables track mode for the tablet, along with the trailing-edge normally used. This lets the user draw smooth lines made of many points. If changing from the draw function to a different function, then etch-sketch restores the tablet mode to trailing-edge only.

(4) After handling changes in its current function, the main section of etch-sketch has select cases on each of the available puck-selected functions. Most select cases contain code to carry out the action you entered. For graphics actions such as draw, point, move, circle, fill, etch-sketch executes graphics display instructions that require coordinates. For change-color actions, etch-sketch executes set-parameter instructions that require color numbers. These instructions requiring coordinates and colors reside at known locations in display list memory and etch-sketch must execute self-modification code (as described above) to store the correct parameters in them before execution.

(5) The etch-sketch enters this termination logic to end a picture entry for two reasons: no more available words in Ramtek common space or user pushes the QUIT button. Note that the host application that sends the etch-sketch to the Ramtek must set a word in Ramtek common containing the maximum space allowed for etch-sketch inputs. This way the host sets a limit on the complexity of the picture. In either end-picture case, the etch-sketch display list branches to the quit code. That code changes the local cursor function table as described above in paragraph G-10. So, after running out of allowed space for a picture, or after pushing the QUIT button, the next puck hit causes the Ramtek to notify the host and jump to the end entry point. At that end entry point the display list makes the cursor invisible. It is up to the host program to disable local function status to prevent the etch-sketch from continuing to operate every time someone pushes the puck buttons.

e. Etch-sketch example. This example shows a relatively significant display list. As in the simple display list example, there are three sections: an application to define and compile an etch-sketch display list, use of the off-line linker, and an application to send the etch-sketch display list, wait for user to finish a picture, then read back that picture and save it on disk.

(1) Define and compile. This program contains the calls to define and compile the etch-sketch display list. As part of the definition, it uses the include file shown above to describe the Ramtek common layout.

```
      PROGRAM DRAW4
CC  *****************************************************
CC
CC  THIS PROGRAM CREATES A DISPLAY LIST TO DO ETCH-SKETCHES.
CC  ANOTHER PROGRAM, TRYDRAW4, USES THIS ETCH-SKETCH DL TO
CC  INPUT PICTURES TO THE HOST COMPUTER.  THE USER ENTERS
CC  THE PICTURE ON THE RM, WITHOUT HOST INTERVENTION THEN
CC  ON PUSHING A 'QUIT' PUCK BUTTON, THE HOST WAKES UP AND
CC  READS BACK THE PICTURE.
CC YOU NEED A 4-BUTTON PUCK.  THIS DL LETS YOU SELECT
CC  ETCH-SKETCH FUNCTIONS USING COMBINATIONS
CC  OF ONE OR MORE BUTTONS ON THE PUCK.
CC  IF YOU NUMBER THE BUTTONS AS FOLLOWS:
CC
CC 1 = YELLOW
CC 2 = WHITE
CC 3 = GREEN
CC 4 = BLUE
CC
CC  THEN, THE ETCH-SKETCH DL GIVES YOU THESE FUNCTIONS.
CC
CC  1           = DRAW
CC  2           = MOVE
CC  4           = DOT
CC  1 + 2       = DECREMENT COLOR
CC  1 + 4       = INCREMENT COLOR
CC  1 + 3       = FILL
CC  2 + 3       = CIRCLE
CC
CC  WHEN TRYDRAW4 USES DRAW4 DL, IT MUST SET LOCAL FUNCTION STATE
CC  FOR CURSORS, AND SET LOCAL CURSOR FUNCTION TO JUMP TO
CC  'DRAW4' DL-NAME.
CC
CC  THERE IS AN AREA OF RM COMMON SET ASIDE FOR COMMO BETWEEN
CC  HOST AND THIS DL.   FILE 'DRAW4.DLCMN' CONTAINS CALLS THAT
CC  DEFINE ITS CONTENTS.
CC
CC  ***********************************************************
      CHARACTER * 6 FCT, SYM, COND
      INTEGER * 2 MSG (80)
      INTEGER * 4 IBIT, IR, IP2, IP3
CC
      PARAMETER NLVL = 1
      PARAMETER IDIM = 8
      INTEGER * 2 NPPL (NLVL)
      INTEGER * 2 ICOL_IN (IDIM, NLVL)
CC
      DATA MSG /80 * '0700'X/
CC
      DATA NPPL /3/
      DATA ICOL_IN /'000'X,  ! BLACK
     *              'F00'X,  ! RED
     *              '0F0'X,  ! GREEN
     *              '00F'X,  ! BLUE
     *              'FF0'X,  ! YELLOW
     *              '0FF'X,  ! CYAN
     *              'F0F'X,  ! PURPLE
     *              'FFF'X / ! WHITE
      CALL KRMINIT (1280, 1024, 4, MON)
      CALL KOVRDEF (NLVL, NPPL, ICOL_IN, IDIM)
CC          DRAW AN ELLIPSE IN COLOR 2
      CALL KFLUSH
CC         ASSIGN USES TO THE VARIOUS DL REGISTERS.
      IDLR_TEMP = 0
```

```
          IDLR_FLAG = 1
          IDLR_STXY = 2
          IDLR_IPTR = 4
          IDLR_MPTR = 3
          IDLR_TEST = 5
          IDLR_CURS = 5
          IDLR_TAB  = 8
          IDLR_COPG = 11
          ITABNUM = 0
          ICONTEXT = 0
CC              OPEN THE DISPLAY LIST
          CALL DL_LISTING (.TRUE., .TRUE.)
CC              OPEN 'DRAW4' DL.
          CALL DL_OPEN ('ODLDATA:DRAW4')
CC              DEFINE COMMON BLOCK 'C1'.
          INCLUDE '[]DRAW3.DLCMN/NOLIST'
CC                RETRIEVE THE CURRENT NUMBER OF ENTRIES, AND
CC                THE MAX REQUESTED NO. OF ENTRIES.
          CALL DL_MEM ('LDLR', IDLR_IPTR, 'IPTR', )
                                                ! CUR # OF TRIPLES IN BUF
          CALL DL_MEM ('LDLR', IDLR_MPTR, 'MPTR', )
CC                QUIT IF CURRENT EQUALS MAX, WITH
CC                A LOCAL JUMP TO THE STOP SECTION.
          CALL DL_R2R ('SUBDLR', 0, IDLR_TEST, IDLR_IPTR, IDLR_MPTR)
          CALL DL_JDLR (IDLR_TEST, 0, 'EQ', 'Z_STOP', )
CC                 THERE ARE THREE WORDS PER ENTRY.
CC                 CONVERT FROM ENTRY COUNT TO WORD COUNT
CC                 TO BYTE COUNT.
          CALL DL_R2R ('MULDLR', 1, IDLR_IPTR, 3, 0)
          CALL DL_R2R ('MULDLR', 1, IDLR_IPTR, 2, 0)
CC                  GET THE STARTING BYTE ADDR IN RM COMMON FOR THE
CC                  BUFFER TO CONTAIN THESE ENTRIES, AND
CC                  INCREMENT BY THE DISPLACEMENT TO THE ADDR FOR
CC                  THE NEXT ENTRY.
          CALL DL_SDLRMEM ( IDLR_STXY, 'STAT_XY', )
          CALL DL_R2R ('ADLR', 0, 2, IDLR_STXY, IDLR_IPTR)
                                                     ! DISPLACEMENT ADDR
CC                  LOAD INTO THREE REGISTERS THE CURRENT
CC                  CURSOR X, Y, AND STATUS.
          CALL DL_PLDLR (IDLR_CURS, 'CURS', ITABNUM)
CC                  LOAD INTO THREE REGISTERS THE CURRENT
CC                  TABLET X, Y, AND STATUS.
          CALL DL_PLDLR (IDLR_TAB, 'TAB', ITABNUM)
CC                  LOAD INTO TWO REGISTERS THE CURRENT
CC                  GLOBAL CURRENT-OPERATING-POINT.
          CALL DL_PLDLR (IDLR_COPG, 'COPG', 0)
CC                  COPY TABLET STATUS TO FLAG REGISTER, AND
CC                  RETAIN THE LOW FOUR BITS, FOR THE FOUR PUCK
CC                  BUTTONS.
          CALL DL_R2R  ('ORDLR', 0, IDLR_FLAG,IDLR_TAB+2,IDLR_TAB+2)
          CALL DL_R2R ('ANDDLR', 1, IDLR_FLAG, '000F'X, )
CC                   IF NO  PUCK BUTTONS SET, RETURN.
          CALL DL_IF   (1, IDLR_FLAG, 'EQ', 0)
CC                  DO NOTHING BUT EXIT.
```

```
           CALL KRETDL
       CALL DL_ENDIF
CC                 IF LAST TABLET BUTTON RELEASED,
CC                 PROCESS TRAILING-EDGE CNDX.
       CALL DL_JDLR (IDLR_TAB+2, 14, 'SET', 'Z_TE', )
CC                  IF FIRST TABLET BUTTON PUSHED,
CC                  PROCESS LEADING-EDGE CNDX.
       CALL DL_JDLR (IDLR_TAB+2, 13, 'SET', 'Z_EN', )
CC                  IF JUST TRACK-MODE FROM TABLET,
CC                  PROCESS TRACK-MODE CNDX.
       CALI DL_JDLR (IDLR_TAB+2,  0, 'GOTO', 'Z_TK', )
CC
CC
 1030  CALL DL_LABEL ('Z_TE', )
CC                 LAST BUTTON RELEASED.
CC                 NO SPECIAL PROCESSING. JUMP TO NORMAL CODE.
       CALL DL_JDLR (IDLR_TAB+2,  0, 'GOTO', 'Z_RUN', )
CC
 1032  CALL DL_LABEL ('Z_EN', )
CC                 FIRST BUTTON PRESSED DOWN.
CC                 CAN'T DO ANYTHING WITH THIS INFO, SO
CC                 EXIT THE DL.
       CALL KRETDL
CC
 1033  CALL DL_LABEL ('Z_TK', )
CC                 TRACK MODE ONLY.
CC                 NO SPECIAL PROCESSING. JUMP TO NORMAL CODE.
       CALL DL_JDLR (IDLR_TEMP, 0, 'GOTO', 'Z_RUN', )
CC
 1035  CALL DL_LABEL ('Z_RUN', )
CC                 COMPARE THE CURRENT FUNCTION IN THE FLAG
CC                 REGISTER WITH THE PREVIOUS FUNCTION.
CC                 IF THE SAME, JUMP TO 'CONTINUE' CODE.
       CALL DL_MEM ('LDLR', IDLR_TEMP, 'IFUNC', )
       CALL DL_MEM ('LDLR', IDLR_LAST, 'IFUNC', )
       CALL DL_R2R ('SUBDLR', 0, IDLR_TEMP, IDLR_TEMP, IDLR_FLAG)
       CALL DL_JDLR (IDLR_TEMP, 0, 'EQ', 'Z_CONT', )
CC
 1040  CALL DL_LABEL ('Z_CHANGE', )
CC                SAVE NEW DL FUNCTION IN RM COMMON.
       CALL DL_MEM ('STDLR', IDLR_FLAG, 'IFUNC', )
CC                IF LAST OPERATION WAS A RUN, THEN CHANGE BACK
CC                TO TE-ONLY ON TABLET.
       CALL DL_IF   (1, IDLR_LAST, 'EQ', 1)
CC              SET THE TABLET TO INTERRUPT IN TWO CASES:
CC              TE = WHEN USER RELEASES THE LAST OF ONE OR MORE
CC                   BUTTONS THAT WERE DEPRESSED.
           ISTAT = '4000'X + '0100'X + + '0F'X
                                        ! (TE+TK) + TD + (F3+F2+F1+F0)
           CALL KWTABST ( ITABNUM, ISTAT, 4, 4)
       CALL DL_ENDIF
CC
CC
CC                     SELECT CASE -- CHANGE TO MOVE
```

```
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 2)
           CALL KWCTEXT (0, 20, 4, 'MOVE', 4)
       CALL DL_ENDIF
CC
CC                      SELECT CASE -- CHANGE TO DRAW
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 1)
           CALL KWCTEXT (0, 20, 4, 'DRAW', 4)
CC               SET THE TABLET TO INTERRUPT IN TWO CASES:
CC               TK = WHEN USER HOLDS DOWN AT LEAST ONE BUTTON AND
CC                    MOVES THE PUCK.
CC               TE = WHEN USER RELEASES THE LAST OF ONE OR MORE
CC                    BUTTONS THAT WERE DEPRESSED.
           ISTAT = '5000'X + '0100'X + + '0F'X
                                        ! (TE+TK) + TD + (F3+F2+F1+F0)
       CALL KWTABST ( ITABNUM, ISTAT, 4, 4)
       CALL DL_ENDIF
CC
CC                      SELECT CASE -- CHANGE TO POINT
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 8)
           CALL KWCTEXT (0, 20, 4, 'DOT ', 4)
       CALL DL_ENDIF
CC
CC                      SELECT CASE -- CHANGE TO CIRCLE
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 6)
           CALL KWCTEXT (0, 20, 4, 'CIRC', 4)
       CALL DL_ENDIF
CC
CC                      SELECT CASE -- CHANGE TO FILL
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 5)
           CALL KWCTEXT (0, 20, 4, 'FILL', 4)
       CALL DL_ENDIF
CC
CC                      SELECT CASE -- CHANGE TO QUIT
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 4)
           CALL KWCTEXT (0, 20, 4, 'QUIT', 4)
       CALL DL_ENDIF
CC
 1100  CALL DL_LABEL ('Z_CONT', )
CC
CC   COMPUTED GOTO ON THE FOUR PUCK BUTTONS IN THE FLAG REGISTER.
CC
CC
CC                      SELECT CASE -- INCREMENT COLOR
CC                      LOAD THE CURRENT COLOR INTO TEMP REGISTER,
CC                      INCREMENT BY ONE. THEN JUMP TO COLOR-SET
CC                      CODE.
       CALL DL_IF   (1, IDLR_FLAG, 'EQ', 9)
           CALL DL_MEM ('LDLR',  IDLR_TEMP, 'FGCOL', )
           CALL DL_IDLR (0, IDLR_TEMP)
CC
CC                          SAVE NEW COLOR IN COMMON, THEN
CC                          MODIFY THE SUBSEQUENT DISPLAY-LIST
CC                          SET COMMAND TO CONTAIN THE NEW COLOR
CC                          VALUE.
```

```
            CALL DL_MEM ('STDLR', IDLR_TEMP, 'FGCOL', )
            CALL DL_MEM ('STDLR', IDLR_TEMP, 'FGC+', )
            CALL KOCOLOR (1)
            CALL DL_LABEL ('FGC+', -1)
            CALL KWCTEXT (0, 20, 4, 'COL+', 4)
CC                  MOVE BACK TO THE CURSOR POSITION FOR THIS
CC                  ENTRY BY STORING IT FROM REGISTERS INTO
CC                  THIS DL CODE, AS THE TWO COP WORDS IN  THE SET
CC                  INSTRUCTION RESULTING FROM 'KMOVEA'.
            CALL DL_MEM ('STDLR', IDLR_CURS, 'COLMOVEX+', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COLMOVEY+', )
            CALL KMOVEA (0, 0)
            CALL DL_LABEL ('COLMOVEX+', -2)
            CALL DL_LABEL ('COLMOVEY+', -1)
CC                  PUT THE CURRENT COLOR INTO THE
CC                  REGISTER TO BE SAVED AS SECOND WORD OF THE
CC                  ENTRY TRIPLE... USUALLY RESERVED FOR X COORD.
            CALL DL_MEM ('LDLR', IDLR_CURS, 'FGCOL', )
        CALL DL_ENDIF
CC
CC                    SELECT CASE -- DECREMENT COLOR
CC                    LOAD THE CURRENT COLOR INTO TEMP REGISTER,
CC                    DECREMENT BY ONE. THEN JUMP TO COLOR-SET
CC                    CODE.
        CALL DL_IF   (1, IDLR_FLAG, 'EQ', 3)
            CALL DL_MEM ('LDLR',  IDLR_TEMP, 'FGCOL', )
            CALL DL_DDLR (0, IDLR_TEMP)
CC
CC                       SAVE NEW COLOR IN COMMON, THEN
CC                       MODIFY THE SUBSEQUENT DISPLAY-LIST
CC                       SET COMMAND TO CONTAIN THE NEW COLOR
CC                       VALUE.
            CALL DL_MEM ('STDLR', IDLR_TEMP, 'FGCOL', )
            CALL DL_MEM ('STDLR', IDLR_TEMP, 'FGC-', )
            CALL KOCOLOR (1)
            CALL DL_LABEL ('FGC-', -1)
            CALL KWCTEXT (0, 20, 4, 'COL-', 4)
CC                  MOVE BACK TO THE CURSOR POSITION FOR THIS
CC                  ENTRY BY STORING IT FROM REGISTERS INTO
CC                  THIS DL CODE, AS THE TWO COP WORDS IN  THE SET
CC                  INSTRUCTION RESULTING FROM 'KMOVEA'.
            CALL DL_MEM ('STDLR', IDLR_CURS, 'COLMOVEX-', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COLMOVEY-', )
            CALL KMOVEA (0, 0)
            CALL DL_LABEL ('COLMOVEX-', -2)
            CALL DL_LABEL ('COLMOVEY-', -1)
CC                  PUT THE CURRENT COLOR INTO THE
CC                  REGISTER TO BE SAVED AS SECOND WORD OF THE
CC                  ENTRY TRIPLE... USUALLY RESERVED FOR X COORD.
            CALL DL_MEM ('LDLR', IDLR_CURS, 'FGCOL', )
        CALL DL_ENDIF
CC                  SELECT CASE -- MOVE.
CC                  MOVE TO THE CURSOR POSITION FOR THIS
CC                  ENTRY BY STORING IT FROM REGISTERS INTO
CC                THIS DL CODE, AS THE TWO COP WORDS IN  THE SET
CC                INSTRUCTION RESULTING FROM 'KMOVEA'.
      CALL DL_IF   (1, IDLR_FLAG, 'EQ', 2)
          CALL DL_MEM ('STDLR', IDLR_CURS, 'MOVEX', )
          CALL DL_MEM ('STDLR', IDLR_CURS+1, 'MOVEY', )
          CALL KMOVEA (0, 0)
          CALL DL_LABEL ('MOVEX', -2)
          CALL DL_LABEL ('MOVEY', -1)
CC              SAVE THIS CURSOR XY  IN COMMON FOR LATER USE
CC              AS COP.
          CALL DL_MEM ('STDLR', IDLR_CURS,   'COPX', )
          CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COPY', )
      CALL DL_ENDIF
CC
CC               SELECT CASE -- DRAW
CC               LOAD THE SAVED COP FROM COMMON INTO REGISTERS.
      CALL DL_IF   (1, IDLR_FLAG, 'EQ', 1)
          CALL DL_MEM ('LDLR',  IDLR_COPG,   'COPX', )
          CALL DL_MEM ('LDLR',  IDLR_COPG+1, 'COPY', )
CC                MOVE TO THE CURRENT OPERATION POSITION FOR LAST
CC                ENTRY BY STORING IT FROM REGISTERS INTO
CC                THIS DL CODE, AS THE TWO COP WORDS IN  THE SET
CC                INSTRUCTION RESULTING FROM 'KMOVEA'.
          CALL DL_MEM ('STDLR', IDLR_COPG, 'DRAW1X', )
          CALL DL_MEM ('STDLR', IDLR_COPG+1, 'DRAW1Y', )
          CALL KMOVEA (0, 0)
          CALL DL_LABEL ('DRAW1X', -2)
          CALL DL_LABEL ('DRAW1Y', -1)
CC                DRAW TO THE CURSOR POSITION FOR THIS
CC                ENTRY BY STORING IT FROM REGISTERS INTO
CC                THIS DL CODE, AS THE TWO COP WORDS IN  THE
CC                WRITE-VECTOR-LINKED
CC                INSTRUCTION RESULTING FROM 'KLADRAW'.
          CALL DL_MEM ('STDLR', IDLR_CURS, 'DRAW2X', )
          CALL DL_MEM ('STDLR', IDLR_CURS+1, 'DRAW2Y', )
          CALL KLADRAW (0, 0)
          CALL DL_LABEL ('DRAW2X', -2)
          CALL DL_LABEL ('DRAW2Y', -1)
CC              SAVE THIS CURSOR XY  IN COMMON FOR LATER USE
CC              AS COP.
          CALL DL_MEM ('STDLR', IDLR_CURS,   'COPX', )
          CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COPY', )
      CALL DL_ENDIF
CC
CC                SELECT CASE -- POINT
CC                DRAW A POINT AT THE CURSOR POSITION FOR THIS
CC                ENTRY BY STORING IT FROM REGISTERS INTO
CC                THIS DL CODE, AS THE TWO COP WORDS IN  THE
CC                PLOT-POINT INSTRUCTION RESULTING FROM 'KDOT'.
      CALL DL_IF   (1, IDLR_FLAG, 'EQ', 8)
          CALL DL_MEM ('STDLR', IDLR_CURS, 'DOTX', )
          CALL DL_MEM ('STDLR', IDLR_CURS+1, 'DOTY', )
          CALL KDOT (0, 0)
          CALL DL_LABEL ('DOTX', -2)
```

```
            CALL DL_LABEL ('DOTY', -1)
CC                 SAVE THIS CURSOR XY  IN COMMON FOR LATER USE
CC                 AS COP.
            CALL DL_MEM ('STDLR', IDLR_CURS,   'COPX', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COPY', )
        CALL DL_ENDIF
CC
CC                SELECT CASE -- CIRCLE
CC                KLUGE CODE -- VERY SORRY ABOUT IT.
CC                DERIVE RADIUS FOR CIRCLE FROM THE ABS DIFF
CC                BETWEEN LAST COP AND CURRENT CURSOR X/Y.
CC                THEN, USE CURSOR X/Y AS CENTER.
        CALL DL_IF   (1, IDLR_FLAG, 'EQ', 6)
            CALL DL_MEM ('LDLR',  IDLR_COPG,   'COPX', )
            CALL DL_MEM ('LDLR',  IDLR_COPG+1, 'COPY', )
            CALL DL_R2R ('SUBDLR', 0,IDLR_TEMP,IDLR_COPG,IDLR_CURS)
CC                WATCH OUT FOR NEGATIVE RADIUS.
CC                IF POSITIVE, JUMP AROUND TO CIRCLE END.
            CALL DL_IF   (1, IDLR_TEMP, 'LT', 0)
CC                 MULTIPLY NEGATIVE RADIUS BY MINUS ONE.
                CALL DL_R2R ('MULDLR', 1, IDLR_TEMP, -1, )
            CALL DL_ENDIF
CC
CC                   SELF-MODIFYING CODE AGAIN.
CC                   DRAW A CIRCLE AT THE CURSOR POSITION FOR THIS
CC                   ENTRY, HAVING THE CALCULATED RADIUS,
CC                   BY STORING X/Y AND RADIUS IT FROM THREE
CC                   REGISTERS INTO THIS DL CODE, AS THE TWO
CC                   CENTER X/Y WORDS AND THE RADIUS WORD IN  THE
CC                   CIRCLE INSTRUCTION, RESULTING FROM 'KCIRCLE'.
            CALL DL_MEM ('STDLR', IDLR_CURS,   'CIRCX', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'CIRCY', )
            CALL DL_MEM ('STDLR', IDLR_TEMP,   'CIRCR', )
            CALL KCIRCLE (0, 0, 0)
            CALL DL_LABEL ('CIRCX', -3)
            CALL DL_LABEL ('CIRCY', -2)
            CALL DL_LABEL ('CIRCR', -1)
CC                 SAVE THIS CURSOR XY  IN COMMON FOR LATER USE
CC                 AS COP.
            CALL DL_MEM ('STDLR', IDLR_CURS,   'COPX', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COPY', )
        CALL DL_ENDIF
CC
CC                   SELECT CASE -- FILL
CC                   FILL AN AREA SURROUNDING THE CURSOR POSITION
CC                   FOR THIS ENTRY WHILE COLOR IS CONSTANT,
CC                   BY STORING IT FROM REGISTERS INTO
CC                   THIS DL CODE, AS THE TWO COP WORDS IN  THE
CC                   FILL INSTRUCTION RESULTING FROM 'KFILL'.
        CALL DL_IF   (1, IDLR_FLAG, 'EQ', 5)
            CALL DL_MEM ('STDLR', IDLR_CURS,   'FILLX', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'FILLY', )
            CALL KFILL (1, 0, 0)
            CALL DL_LABEL ('FILLX', -4)
            CALL DL_LABEL ('FILLY', -3)
CC                 SAVE THIS CURSOR XY  IN COMMON FOR LATER USE
CC                 AS COP.
            CALL DL_MEM ('STDLR', IDLR_CURS,   'COPX', )
            CALL DL_MEM ('STDLR', IDLR_CURS+1, 'COPY', )
        CALL DL_ENDIF
CC
        CALL DL_IF   (1, IDLR_FLAG, 'EQ', 4)
CC                SELECT CASE -- QUIT (STOP)
CC                CLEAR THE LOCAL CURSOR FUNCTION NOW IN USE,
CC                AND SET A DIFFERENT LOCAL CURSOR FUNCTION TO
CC                JUMP ON INTERRUPT TO A DIFFERENT ENTRY POINT IN
CC                THIS DISPLAY LIST, WITH THE H-BIT SET TO
CC                INFORM THE HOST COMPUTER THAT SHOULD BE
CC                WAITING FOR AN INTERRUPT FROM THE RAMTEK.
CC                THIS IS THE MECHANISM TO SYNCHRONIZE THE
CC                HOST AND RAMTEK.
            CALL DL_CLCF (ITABNUM)
            CALL DL_SLCF ( 1, ITABNUM, ICONTEXT, , 'DRAW4_END')
        CALL DL_ENDIF
CC
 1200   CALL DL_LABEL ('Z_STORE', )
CC             STORE SOFTWARE.  COPIES ONE TRIPLE FROM THREE
CC             REGISTERS INTO THE COMMON AREA, STARTING  AT THE
CC             BYTE ADDRESS SPECIFIED BY A FOURTH REGISTER.
CC             USES THE REGISTER-TO-MEMORY STORE DLR IMMEDIATE
CC             INSTRUCTION.  BUMPS THE FOURTH (ADDRESS) REGISTER
CC             BY TWO BYTES BETWEEN EACH STORE OPERATION.
        CALL DL_MEMI ('STDLR', IDLR_FLAG, IDLR_STXY)
        CALL DL_IDLR (1, IDLR_STXY)
        CALL DL_MEMI ('STDLR', IDLR_CURS, IDLR_STXY)
        CALL DL_IDLR (1, IDLR_STXY)
        CALL DL_MEMI ('STDLR', IDLR_CURS+1, IDLR_STXY)
        CALL DL_MEM ('LDLR', IDLR_IPTR, 'IPTR', )
                                              ! GET OLD COUNT OF TRIPLES
CC                INCREMENT AND SAVE IN COMMON THE COUNT OF THE
CC                NUMBER OF TRIPLES.
        CALL DL_IDLR (0, IDLR_IPTR)                    ! BUMP COUNT
        CALL DL_MEM ('STDLR', IDLR_IPTR, 'IPTR', ) ! SAVE NEW COUNT
        CALL KRETDL
 2000   CALL DL_LABEL ('Z_STOP', )
CC                SELECT CASE -- QUIT (STOP)
CC                CLEAR THE LOCAL CURSOR FUNCTION NOW IN USE,
CC                AND SET A DIFFERENT LOCAL CURSOR FUNCTION TO
CC                JUMP ON INTERRUPT TO A DIFFERENT ENTRY POINT IN
CC                THIS DISPLAY LIST, WITH THE H-BIT SET TO
CC                INFORM THE HOST COMPUTER.
        CALL DL_CLCF (ITABNUM)
        CALL DL_SLCF ( 1, ITABNUM, ICONTEXT, , 'DRAW4_END')
        CALL KRETDL
CC
 5000   CALL DL_ENTRY ('DRAW4_END', )
CC                  INTERRUPT COMES HERE WHEN H-BIT IS SET.
CC                  TURN OFF THE CURSOR.
```

```
      CALL KWCURS ( 0, 0, 640, 512)
CC                 IMPLIED KRETDL FROM THE CLOSE.
      CALL DL_CLOSE
 9990 CONTINUE
      STOP
      END
```

(2) Off-line linker. Next, to link the three display lists you run the off-line display-list linker program as follows.

```
$ RUN OFLIKDL:LINKER
MAKE list of DL's to link
DL-name, or <CR> to end ODLDATA:DRAAW4
DL-name, or <CR> to end <cr>
ENTER DL-EXE file name ODLDATA:DRAW4
output a listing? [Y/N] [N] Y
DL_DLLINK: link phase
DL_DLLINK: commons  phase
DL_DLLINK: positioning phase
DL_DLLINK: segment phase
DL_LINK: save-EXE  phase
DL_DLPREP: external  0047 4100 8000
DL_DLPREP: external  0039 4100 8094
LOADED DLS
FORTRAN STOP
```

(3) Send and wait for interrupt. This program sends the etch-sketch display list to the Ramtek, defines the same Ramtek common areas for itself, then waits for a tablet read until etch-sketch flags that the user finished a picture. Then, this host program reads back the display list common and saves it on a VAX disk file for later redrawing.

```
      PROGRAM RUNDRAW4
CC   ***************************************************
CC
CC   THIS MODULE TESTS THE DISPLAY LIST NAMED 'DRAW4'.
CC   IT LOADS THE DISPLAY LIST AND THEN ENTERS THE MAIN LOOP OF
CC   THE PROGRAM.
CC
CC   IN ITS MAIN LOOP, THE HOST PROGRAM ASKS YOU FOR A MAX
CC   DRAWING LENGTH, AND THEN SETS THAT VALUE
CC   AND SOME OTHER VARIABLES IN COMMON ON THE
CC   RAMTEK, THEN GOES INTO A WAIT STATE.
CC
CC   ON THE RAMTEK, 'DRAW4' USES INTERRUPTS KEPT SOLELY
CC   ON THE RAMTEK TO ALLOW YOU TO
CC   DRAW PICTURES AND STORE YOUR WORK ON THE RAMTEK
CC   IN A BUFFER IN RM COMMON.
CC   WHEN YOU FINISH, YOU HIT THE QUIT FUNCTION TWICE, WHICH
CC   WILL GENERATE AN INTERRUPT TO THE HOST THAT IS IN A
CC   WAIT STATE.
```

```
CC
CC  BACK ON THE HOST,  AFTER BEING AWAKENED BY AN INTERRUPT FROM
CC  THE 'QUIT' IN 'DRAW4' DISPLAY LIST, THE HOST READS BACK THE
CC  DRAWING YOU ENTERED FROM THE RAMTEK COMMON INTO A BUFFER ON
CC  THE HOST SIDE.  THEN, THIS HOST PROGRAM OPENS A FILE, AND
CC  SAVES YOUR DRAWING ON THAT FILE.  THIS ENDS THE MAIN LOOP, WHICH
CC  REPEATS UNTIL YOU ENTER A MAX DRAWING LENGTH OF ZERO.
CC
CC  *****************************************************
      CHARACTER  * 12 DL_LIST (64)
      CHARACTER * 1 YESNO
      CHARACTER  * 12 DL_SEL
      LOGICAL SEPARATE_LOAD
      LOGICAL ALREADY_IN_RAMTEK
      CHARACTER * 4 MSG1 /'TEST'/
      CHARACTER * 8 MSG2 /'TESTING'/
      CHARACTER * 16 MSG3 /'THIS IS A TEST'/
CC
      REAL * 4 X (5), Y (5)
      INTEGER * 2 QWIN (4)
CC
      PARAMETER NLVL = 1
      PARAMETER IDIM = 16
      INTEGER * 2 NPPL (NLVL)
      INTEGER * 2 ICOL_IN (IDIM, NLVL)
      INTEGER * 2 STAT_XY (3, 1000)
CC
      CHARACTER * 60 OUT_FILE
      LOGICAL INIT_TBUTN
CC
      DATA NPPL /3/
      DATA ICOL_IN /'000'X,  ! BLACK
     *             'F00'X,  ! RED
     *             '0F0'X,  ! GREEN
     *             '00F'X,  ! BLUE
     *             'FF0'X,  ! YELLOW
     *             '0FF'X,  ! CYAN
     *             'F0F'X,  ! PURPLE
     *             'FFF'X,  ! WHITE
     *             '888'X,
     *             'F88'X,
     *             '8F8'X,
     *             '88F'X,
     *             'FF8'X,
     *             '8FF'X,
     *             'F8F'X,
     *             'FFF'X/
CC        ARRAY FOR VIRTUAL LINE-DRAW TEST
CC
      CALL KRMINIT (1280, 1024, 4, MON)
      CALL KQWIN (QWIN)
      CALL KOVRDEF (NLVL, NPPL, ICOL_IN, IDIM)
      CALL KFLUSH
CC      SET TABLET # FOR THIS MONITOR.
```

```
      ITABNUM = 0
CC        DEALLOCATE CONTEXT, THEN REALLOFLIKE IT.
      ICONTEXT = 0
C      CALL DL_DECON (ICONTEXT)
C      CALL DL_ALCON (ICONTEXT)
CC       DISABLE CURSOR INTERRUPTS ON RAMTEK FOR THAT TABLET,
CC       AND CLEAR THE LOCAL CURSOR FUNCTION TABLE FOR THIS
CC       TABLET.
      CALL KSETLCFS (ITABNUM, 1)
      CALL DL_CLCF (ITABNUM)
CC        SET WRITE MASK, AND BACKGROUND AND FOREGROUND COLORS.
      CALL KOVRBUF (1)
      CALL KOCOLOR (2)
      CALL KOBCOLR (1)
CC        SET THE VIDEO ORIGIN TO UPPER LEFT CORNER.
      CALL KVO (0)
CC        DISABLE DL-PACKAGE LIST OUTPUT.
      CALL DL_LISTING (.FALSE., .TRUE.)
CCCCCCCALL DL_LISTING (.TRUE., .TRUE.)
CC        LOAD THE 'DRAW4' DISPLAY LIST FROM FILE TO RAMTEK.
      CALL DL_SEND  ('ODLDATA:DRAW4', NDL, DL_LIST)
CC        USE DL-COMMON CALLS TO DEFINE FOR THE HOST THE
CC        COMMON BLOCK AND ITS VARIABLES USED ON THE RAMTEK.
      INCLUDE '[]DRAW3.DLCMN/NOLIST'
      CALL DL_LAYOUTCMN
CC         ASK USER FOR WHICH DL'S TO CALL.
      INIT_TBUTN = .TRUE.
CC
CC  - - - - - - - - - - -   MAIN LOOP   - - - - - - - - - - - - -
CC
 2000 CONTINUE
CC            GET THE MAX NO. OF ENTRIES FOR PICTURE.
CC            CURRENTLY THERE IS ONLY ROOM FOR 1000.
      WRITE (6, '(A, $)')
     *    ' ENTER max no. of points, or zero to quit '
      READ (5, *, ERR = 2000) MPTR
      IF (NPTS .GT. 1000) GOTO 2000
      IF (MPTR .LE. 0) GOTO 9990
CC
CC
      KNT = 0
 2010 CONTINUE
CC         SET UP INITIAL VARIABLES ON HOST SIDE.
      ICOPX = 0
      ICOPY = 0
      IFGCOL = 1
      IFUNC = 0
      JSAVE = 0
      KNT = KNT + 1
      IPTR = 0
      DO I = 1, MPTR
          DO J = 1, 3
               STAT_XY (J, I) = 0
          ENDDO
      ENDDO
CC          WRITE THOSE VARIABLES TO THE RAMTEK COMMON.
CC                   FUNCTION COMMON VARIABLE WORD HOST  WORD
CC                     CODE    NAME   NAME   OFFSET BUF  XFER
      CALL DL_ACCESSCMN ('WR', 'C1', 'COPX',   0, ICOPX,  1,IERR)
      CALL DL_ACCESSCMN ('WR', 'C1', 'COPY',   0, ICOPY,  1,IERR)
      CALL DL_ACCESSCMN ('WR', 'C1', 'FGCOL',  0, IFGCOL, 1,IERR)
      CALL DL_ACCESSCMN ('WR', 'C1', 'IFUNC',  0, IFUNC,  1,IERR)
      CALL DL_ACCESSCMN ('WR', 'C1', 'JSAVE',  0, JSAVE,  1,IERR)
      CALL DL_ACCESSCMN ('WR', 'C1', 'MPTR',   0, MPTR,   1,IERR)
      CALL DL_ACCESSCMN ('WR', 'C1', 'IPTR',   0, IPTR,   1,IERR)
      CALL DL_ACCESSCMN ('WR','C1','STAT_XY',0, STAT_XY,MPTR*3,
     *    IERR)
D     CALL DL_ADDR              ('C1','STAT_XY',0, IADDR, ISEG,
     *    IERR)
D     WRITE (6, '(A, Z6.4, Z6.4, I6)')
D    *      'IADDR, ISEG, IERR', IADDR, ISEG, IERR
CC            SET THE TABLET TO INTERRUPT IN TWO CASES:
CC            TK = WHEN USER HOLDS DOWN AT LEAST ONE BUTTON AND
CC                 MOVES THE PUCK.
CC            TE = WHEN USER RELEASES THE LAST OF ONE OR MORE
CC                 BUTTONS THAT WERE DEPRESSED.
      ISTAT = '5000'X + '0100'X + + '0F'X
                                       ! (TE+TK) + TD + (F3+F2+F1+F0)
      CALL KWTABST ( ITABNUM, ISTAT, 4, 4)
CC           MAKE THE CURSOR VISIBLE.
      ISTAT = 1
      CALL KWCURS ( ITABNUM, ISTAT, 640, 512)
CC         SET THE FOREGROUND COLOR TO THAT LOADED INTO
CC         RAMTEK COMMON.
      CALL KOCOLOR (IFGCOL + 1)
CC      ENABLE CURSOR INTERRUPTS ON RAMTEK FOR THAT TABLET.
      CALL KSETLCFS (ITABNUM, 2)
CC      SET THE LOCAL CURSOR FUNCTION FOR THIS TABLET
CC      SO THE RAMTEK, ON INTERRUPTS FROM THAT DEVICE,
CC      WILL JUMP TO DL-ENTRY POINT 'DRAW4', AND EXECUTE
CC      THIS DISPLAY LIST.
      IHBIT = 0
      CALL DL_SET_LCF ( IHBIT, ITABNUM, ICONTEXT,  , 'DRAW4')
CC      WAIT UNTIL RAMTEK SENDS BACK AN INTERRUPT.
      CALL KRTABST ( ITABNUM, IX, IY, ISTAT )
CC          READ BACK THE NO. OF PTS ENTERED.
      CALL DL_ACCESSCMN ('RD', 'C1', 'IPTR', 0, IPTR,  1, IERR)
CC         IF THERE ARE POINTS ENTERED, THEN
      IF (IPTR .GT. 0) THEN
CC         READ BACK THE BUFFER OF POINTS.
          CALL DL_ACCESSCMN ('RD','C1','STAT_XY',0,
     *                                  STAT_XY,IPTR*3,IERR)
CC           PREPARE OUTPUT FILE NAME, OPEN FILE, AND
CC           SAVE THE PICTURE ON THE FILE.
 2020     CONTINUE
          WRITE (OUT_FILE, '(A, I3.3, A)') 'TRYDRAW4', KNT, '.DAT'
          LENF = LASTNONB (OUT_FILE)
          WRITE (6, '(A, $)')
     *        ' Enter OUT_FILE, or <CR> = ', OUT_FILE (1 : LENF)
          READ (5, '(Q, A)') LENINP, OUT_FILE
          IF (LENINP .LE. 0) THEN
              WRITE (OUT_FILE,
     *          '(A, I3.3, A)') 'TRYDRAW4', KNT, '.DAT'
          ENDIF
          OPEN (UNIT = 2, NAME = OUT_FILE,
     *       TYPE = 'NEW', ERR = 2020)
CC
          WRITE (6, *) 'TRYDRAW4: IPTR = ', IPTR
          WRITE (2, *) IPTR, QWIN (3), QWIN (4)
          DO I = 1, IPTR
              JBUTN = STAT_XY (1, I)
              CALL DL_TBUTN (INIT_TBUTN, JBUTN, ISINGLE)
              INIT_TBUTN = .FALSE.
D             WRITE (6, '(Z6.4, Z6.4, Z6.4, Z6.4, I6, 1X, I6)')
D    *            (I-1)*2, JBUTN, ISINGLE, (STAT_XY (J, I),
D    *                J = 1, 3)
               WRITE (2, '(Z6.4, I6, I6)')
     *                (STAT_XY (J, I), J = 1, 3)
          ENDDO
          CLOSE (UNIT = 2)
      ELSE
          INIT_TBUTN = .TRUE.
          GO TO 2010
      ENDIF
      GO TO 2000
CC
 9990 CONTINUE
      STOP
      END
```

DISTRIBUTION LIST

	No. Copies
Defense Technical Information Center ATTN: DTIC, TCA Cameron Station Alexandria, VA 22314	2
US Army Library Army Study Documentation and Information Retrieval System (ASDIRS) ANRAL-RS ATTN: ASDIRS Room 1a518, The Pentagon Washington, D.C. 20310	1
US Army TRADOC Analysis Command-WSMR ATTN: ATRC-WSL (Technical Library) White Sands Missile Range, NM 88002-5502	1
US Army TRADOC Analysis Command-FLVN ATTN: ATRC-FOA (Technical Info Center) Fort Leavenworth, KS 66027-5200	1
US Army Combined Arms Research Library (CARL) ATTN: ATZL-SWS-L Fort Leavenworth, KS 66027-5000	1
US Army TRADOC Analysis Command ATTN: ATRC Fort Leavenworth, Kansas 66027-5200 Thru Director TRADOC Analysis Command-FLVN ATTN: ATRC-F Fort Leavenworth, Kansas 66027-5200	1